CLASSIFICATION OF AND RESILIENCE TO CYBER-ATTACKS ON CYBER-PHYSICAL SYSTEMS

A Thesis
Presented to
The Academic Faculty

By

Kevin G. Lyn

In Partial Fulfillment
Of the Requirements for the Degree
Masters of Science in Electrical and Computer Engineering

Georgia Institute of Technology
August 2015

Classification of and Resilience to Cyber-Attacks on Cyber-Physical Systems

Approved by:

Dr. Lee W. Lerner, Co-Advisor
Cyber Technology and Information Security Laboratory
Georgia Institute of Technology

Dr. Thomas R. Collins, Co-Advisor
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Raheem A. Beyah
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Edward J. Coyle
School of Electrical and Computer Engineering
Georgia Institute of Technology

Date Approved: July 23, 2015

To God Almighty

ACKNOWLEDGEMENTS

I would like to thank my research advisor, Dr. Lee W. Lerner, for providing me with the opportunity, resources, and guidance which made this work possible. I am also very grateful for Dr. Michael E. West for providing me with the opportunity to work at the Georgia Tech Research Institute. In addition, I would like to thank my Virginia Tech colleagues who participated in TAIGA experimental design and implementation, namely Christopher J. McCarty and Dr. Cameron D. Patterson. I would also like to thank my co-advisor, Dr. Thomas R. Collins, and my thesis reading committee members, Dr. Raheem A. Beyah and Dr. Edward J. Coyle, for their support with this work. Finally, I am grateful for the support and encouragement of my family and friends, without whom my studies at Georgia Tech would not be possible.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

CPS	Cyber-Physical System
TAIGA	Trustworthy Autonomic Interface Guardian Architecture
HMI	Human-Machine Interface
NIST	U.S. National Institute of Standards and Technology
ICS	Industrial Control System
DoS	Denial-of-Service
FPGA	Field Programmable Gate Array
VAN	Virtual Automation Network
SCADA	Supervisory Control and Data Acquisition
E-Stop	Emergency Stop
SoC	System-on-Chip
HLS	High-Level Synthesis
RTOS	Real-Time Operating System
API	Application Program Interface
UI	User Interface
SerDes	Serializer / Deserializer
FIFO	First-In-First-Out
PRR	Partial Runtime Reconfiguration
HAL	Hardware Abstraction Layer

SUMMARY

The growing connectivity of cyber-physical systems (CPSes) has led to an increased concern over the ability of cyber-attacks to inflict physical damage. Current cybersecurity measures focus on preventing attacks from penetrating control supervisory networks. These reactive techniques, however, are often plagued with vulnerabilities and zero-day exploits. Embedded processors in CPS field devices often possess little security of their own, and are easily exploited once the network is penetrated. In response, researchers at Georgia Tech and Virginia Tech have proposed a Trustworthy Autonomic Interface Guardian Architecture (TAIGA), which monitors communication between the embedded controller and physical process. This autonomic architecture provides the physical process with a last line of defense against cyber-attacks by switching process control to a trusted backup controller if an attack causes a system specification violation.

This thesis focuses on classifying the effects of cyberattacks on embedded controllers, evaluating TAIGA's resilience against these attacks, and determining the applicability of TAIGA to other CPSes. This thesis identifies four possible outcomes of a cyber-attack on a CPS embedded processor. We then evaluate TAIGA's mechanisms to defend against those attack outcomes, and verify TAIGA satisfies the listed trust requirements. Next, we discuss an implementation and the experimental results of TAIGA on a hazardous cargo transportation robot. Then, by making various modifications to the setup configuration, we are able to explore TAIGA's ability to provide security and process protection to other CPSes with varying levels of autonomy or distributed components.

CHAPTER 1

INTRODUCTION

With the promise to improve reliability, efficiency, and capability of physical processes, cyber-physical systems (CPSes) are integrating themselves into nearly every facet of modern society. CPSes are defined as systems that utilize some degree of computing to control or monitor a physical process.

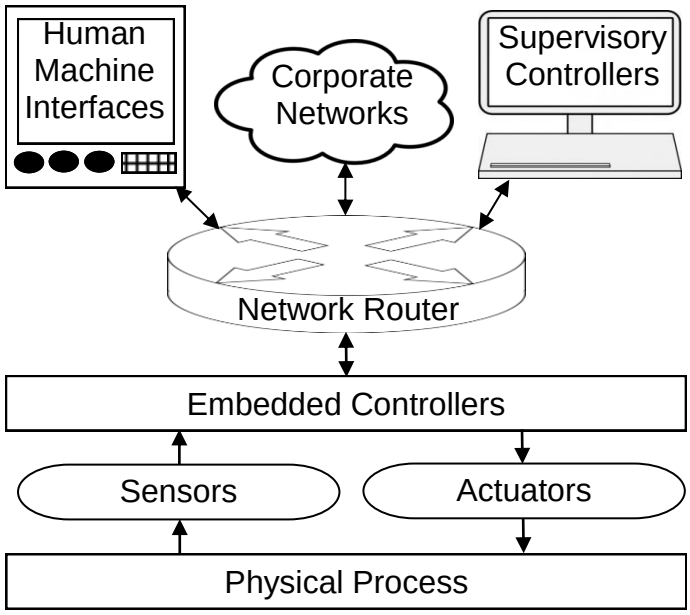

Figure 1: Generic cyber-physical system setup

As shown in Figure 1, CPSes typically involve an embedded process controller, often referred to as a field device, directly connected to the physical process's sensors and actuators. The embedded processor receives instructions from a supervisory network, which includes a variety of components such as network routers, human machine

interfaces (HMIs), supervisory work stations, and external sensors. The supervisory network often permits remote access through corporate networks and internet connectivity. Applications of CPSes can be found in modern consumer electronics, aviation, medical devices, industrial automation, ground transportation, telecommunications, and power systems [1]. The U.S. National Institute of Standards and Technology (NIST) projects advances in CPSes will create entirely new markets with enormous economic impacts, and will be critical for national manufacturing competiveness. The effects of the resultant technologies have the potential to improve the overall quality of human life [2].

As the applications of CPSes continue to increase, so does the potential for cyber-attacks to cause system failures and result in physical damage, environmental impacts, financial losses, or human casualties [3]. In 2000, a disgruntled former contractor for Maroochy Water Services in Australia used radio transmitters in his car to release one million liters of raw sewage into nearby waterways [4]. In 2009, the Stuxnet worm spread through a USB stick is believed to have damaged centrifuges in Iran's Natanz uranium enrichment facility [5]. In December 2014, attackers using a phishing campaign gained access to the network of a German steel mill and inflicted massive damage by improperly shutting down a blast furnace [6]. In July 2015, security experts wirelessly hacked the control systems of a Jeep and were able to override the driver's controls [7].

These and other similar attacks have exposed an urgent need to improve the security of CPSes to prevent them from inflicting physical harm [8]. Most CPS security efforts have focused on protecting the perimeter of supervisory networks controlling the embedded processors. Security measures include typical IT security firewalls, anti-virus

packages, data encryption, access control, and user authentication. Once an attacker penetrates the supervisory network, however, they often find little or no security or validation schemes on the embedded process controller. With the ability to modify the behavior of the embedded processor, the attacker is able to modify the behavior of the physical process [9].

In an effort to improve CPS security, researchers from Georgia Tech and Virginia Tech proposed the Trustworthy Autonomic Interface Guardian Architecture (TAIGA) [10]. In this thesis, we examine the implementation criterion for TAIGA, evaluate its effectiveness, and determine its applicability to other systems.

1.1 Thesis Overview

The remainder of the paper is organized as follows. Chapter 2 discusses CPS adversaries and cyber-attack vectors, categorizes the possible effects of various attacks, and discusses other work related to CPS security. Next, Chapter 3 examines TAIGA's mechanisms for counteracting cyber-attacks and satisfying the trust requirements. Chapter 4 discusses the implementation and experimental results of this scheme on a hazardous cargo transportation robot. In Chapter 5, we make slight modifications to the experimental setup to develop a set of system conditions that determine what benefits TAIGA can provide to other CPSes with varying levels of autonomy and distributed components. The final Chapter provides a summary of the thesis, the impacts of the research, and suggestions for future work.

CHAPTER 2

CYBER-THREATS TO CPSES

Cybersecurity and control design have historically been treated as independent fields, and developed with little consideration for one another [11]. Traditional cybersecurity fails to account for the interdependencies between the cyber and physical systems [12]. Likewise, control design monitors for faults in the physical processes, but often ignores the possibilities of cyber-threats [13]. The increasing demands for CPSes to offer remote access and monitoring resulted in a makeshift mix between the two fields. Traditional IT security technologies, previously designed to protect the exchange of information between cyber components, are being deployed to protect physical processes on CPS networks [14]. Due to the scale and complexity of many CPS networks, preventing infiltration is far from guaranteed. Furthermore, example exploits such as Heartbleed have demonstrated that even supposedly secure communication protocols may be compromised by zero-day exploits and implementation flaws. As a result, traditional cyber perimeter defenses are often considered to be a reactive to newly discovered threats, and cannot promise protection against unknown exploits [10].

In addition, these security techniques assume trust and security in the embedded field devices controlling the physical process. They assume these devices are "protected" by the network, and thus will not be victim to Trojans, or code modification [15]. These devices, however, often include little security measures of their own. Furthermore, the devices are often built using custom or proprietary protocols, hardware description

languages, and third-party cores. These sources can easily hide unintentional or intentional Trojan behaviors and vulnerabilities [16].

Cárdenas' adversary model lists multiple possible CPS cyber-attackers: cybercriminals, disgruntled employees, terrorist groups, and nation states [17]. While each group can inflict harm to CPSes, their methods of attacks vary. For example, cybercriminals often target any available computer in search of sensitive financial information, or to generate spam. Even though their malicious programs are not designed to target CPSes, cybercriminal malware may infect control systems using common IT components, and produce unintended side-effects. For example, in 2003 the Slammer worm targeting IT components shut down the safety protection system at the Davis-Besse nuclear power station in Ohio for several hours [18]. In contrast to cyber criminals, disgruntled employees represent a different risk to CPSes. In addition to possessing insider knowledge of the process and its controllers, current employees often have authorized system access. This allows them to bypass many perimeter security measures, and launch system-specific attacks. Likewise, terrorist and organized criminal groups may also create system-specific attacks directed at CPSes. Groups may aim to cause maximum physical damage as quickly as possible, or hold the system hostage and demand a ransom. Cyber-attacks are incentivizing to terrorist organizations since they are lower-risk and easier to execute than physical attacks. These incentives may also encourage nation states to invest in CPS attack technologies. Because of their available resources, nation state created attacks are often highly sophisticated, and may be designed to coordinate with physical attacks [19].

2.1 Cyber-Attack Outcomes on Process Controllers

Cyber-attacks may take the form of network intrusions, Trojans, malware, malicious updates, code injection, memory modification, denial-of-service attacks, real-time hijacking, or similar attack-vectors [20]. Such attacks may originate from a remote network, be launched from the supervisory network, placed directly in the process controller itself, or hidden in embedded sensor logic. Many of these attacks include stealth components to report the process as operating normally for the duration of the attack. Yet despite the numerous CPS attack vectors, most attacks affect the process controller in similar fashions [21]. Regardless of the attack vector, the effect on the compromised controller can be categorized into four main effect groups as summarized in Table 1.

Table 1: Cyber-attack outcomes on process controllers

Effect on Embedded Controller	**Example Attack Vectors**
Controller Non-Functional - controller stops functioning entirely	Malware, malicious updates
Controller Incorrect - controller issues incorrect, invalid, or shuffled commands	Network intrusions, Trojan logic, code injection, memory modification
Controller DoS - controller issues many unnecessary or surreptitious commands	Denial-of-service
Malicious Controller - controller issues intelligent commands to degrade or damage the process	Real-time hijacking, specially tailored malware

The first possible attack outcome, Controller Non-Functional, occurs when the embedded controller stops functioning entirely. This can occur as a result of simple low-

knowledge malware or malicious updates which delete data or corrupt programs. Processes lacking physical safety interlocks may continue to operate unsafely until the process is damaged or destroyed.

The second attack outcome, Controller Incorrect, arises when the controller still appears to function, however it exhibits occasional erroneous or surreptitious behavior. This type of behavior is often the result of low-knowledge network intrusions, difficult to detect Trojan logic, code injection, or memory injection attacks [22]. The process may exhibit strange behavior and reduced efficiency, and may eventually result in physical process damage.

A third attack outcome, Controller DoS, results when the controller attempts to act on or issues numerous useless commands. Such an outcome is typically the work of a denial-of-service attack. The injected commands monopolize computing and communication resources, delaying or overwriting valid commands. The command disruption may result in physical damage.

The final outcome of an attack on the embedded processor, Malicious Controller, occurs when a knowledgeable attacker commandeers process control. This may be the result of real-time hijacking or specially tailored malware. In either case, the attacker is usually knowledgeable enough about the system to quickly drive the process to destruction.

2.2 Related Work

As awareness of CPS security issues continues to grow, so does research into the field. Because CPS security is a complex issue, it cannot be solved with a single simple

solution, but rather requires research into solving a broad field of challenges [23]. As a result, studies in the field often involve very different focuses, such as supervisory control and data acquisition (SCADA) security, secure communication protocols, specialized device firewalls, resilient control algorithms, intrusion detections techniques, authentication schemes, and hardware configurations [12]. While the discussion of CPS security techniques in this section is by no means exhaustive, it provides an insight into the types of efforts working towards securing CPSes.

For example, some researchers have focused on developing control system models that account for the possibility of cyber-attacks. Cárdenas proposed a resilient control system methodology for systems at risk of denial-of-service (DoS) or deception attacks against their sensors. The framework adds several adversary terms into the traditional state-space model, and uses knowledge of the physical system as a method of detecting cyber-attacks. The technique provides criteria under which the system may operate safely for the duration of the attacks [24] [25]. In some cases, developing the complex physical system models required is not always possible. In response, the proposed Cross Correlation technique was developed to also handle injection attacks without the need for a precise system model. The technique relies on trusted sensors to provide sufficient information to determine if an injection attack is occurring [26] [27]. Both designs, however, fail to account for the possibility of an attack directly targeting the process controller in which the well-designed control algorithm is modified.

Other control-theory examples include Sha's Simplex and L1Simplex control architectures, which utilize a decision module to select the best output from multiple process controllers. The setups are designed to correct for controller errors and system

faults while allowing for real-time controller updates [28] [29]. The architecture, however, provides limited protection against cyber-threats as "the availability of the application controllers can be manually altered by the user through the user interface [30]." Attackers could modify all controllers, forcing the architecture to execute malicious commands.

Other studies have focused on developing verification and testing techniques to formally analyze CPS security. For example, Cheminod presented an industrial control system (ICS) cyber analysis tool written in Prolog. The tool uses various details of the system's network connection and known attack vectors to produce a cybersecurity model of the system which is used to analyze possible attacks [31]. Although useful, this software tool cannot detect previously unknown exploits not included in the security model.

Another set of studies concentrate on preventing cyber-attack penetration through secure communication protocols designed to run large scale control networks. For instance, the Virtual Automation Networks (VAN) solution attempts to define network communication protocol in such a way that undesirable communication between devices can be detected and prevented [32]. Another example is The Scalable Group Key Management Protocol (SGKMP), which uses key management protocols specially designed to scale to large networks with many devices operating under limited bandwidth [33]. Similarly, the Advanced SCADA Key-Management Architecture (ASKMA) is designed to reduce the required computational resources required for field devices to store and use network keys [34].

Other research groups, in comparison, focus on designing trusted hardware or software architectures that protect a device's critical functions even if other components are compromised. For example, the built-in self-authentication (BISA) technique uses digitally signed filler cells to prevent and detect Trojans from occupying unused spaces in critical components [35]. Another approach is ARM's TrustZone architecture, which partitions applications into either the normal world (NWorld) or the secure world (SWorld) resource groups based on their level of trust [36]. Another such technique uses a low-cost, tamper-resistant, trusted physical component known as a Trusted Platform Module (TPM). The TPM is often added as a means to secure cryptographic key functionality, endorsement services, critical data storage, and integrity measurements [37] [38]. TPMs can be used as the root-of-trust for systems such as Microsoft's Next-Generation Secure Computing Base (NGSCB), which uses the TPM to fingerprint the system configuration on boot-up, and distribute trust [39].

Further security approaches employ additional hardware to mitigate or detect cyber-attacks. Barbareschi presents the possibility of using configurable field programmable gate arrays (FPGAs) to change the configuration of the control network to respond to cyber-attacks. CPS devices could automatically reconfigure to replace the functionality of lost devices, or change encryption techniques to dynamically react to shifting demands between performance and confidentiality [40]. Likewise, the Morph Onion-encryption Replication PRR HAL (MORPH) scheme uses FPGAs to constantly shift hardware component locations, making it difficult for Trojans to target specific functions [41]. In similar applications, FPGA lattice-based cryptography is being developed to provide low-resource yet high performance FPGA device security [42]. Another CPS security

approach involves using a sensor array known as Intelligent Checkers to monitor the system's health. The Intelligent Checkers safeguard themselves from cyber-attacks by offering only one-way outgoing communication. The Checkers help protect against stealth attacks by raising an audible and visual alarm to alert system operators if a system anomaly is detected [43]. Intelligent Checkers, however, are reliant on an external operator with the ability to both react quick enough, and to have the physical means to restore the affected system.

2.2.1 TAIGA Prior Work

Although the security measures above represent important developments for CPS security, most measures still possess vulnerabilities that can lead to eventual physical harm as the result of cyber-attacks. The TAIGA architecture uses an additional trusted hardware component to serve as a last line of defense to prevent physical harm. In this section we briefly discuss other works directly related to the development of TAIGA.

The notion of using a hardware-implemented and formally verified controller to monitor and enforce application-specific logic is presented in [44]. The work introduces the high-level design flow for atomic guarded rule enforces. It also utilizes a Bluespec high-level language to provide abstractions for defining logic guards. Next, the use of the TAIGA prediction module was presented in [21]. In [45], the specification guards, switching logic, and back-up controller are implemented using high-level synthesis for C code. A method for formally analyzing and verifying the source code using Frama-C is presented. Later, TAIGA is discussed in an ICS setting for the first time in [46] and [47]. These works presented a secure method for updating TAIGA specification guards to

account for the plant's actuators changing over time. TAIGA is then implemented onto a Zedboard, and an analysis of the required resources and added latency is provided.

Chapter 2 of [10] presents the five trust requirements a device must satisfy to be considered trustworthy. It then uses the requirements to compare TAIGA with other trust-based architectures to illustrate the trust advantages of TAIGA. The work also provides some preliminary experimental results using the Jonny-Five humanoid robot which later evolved into the basis for this thesis.

CHAPTER 3

TAIGA

The Trustworthy Autonomic Interface Guardian Architecture (TAIGA) is designed to guard against the four controller attack outcomes. TAIGA can therefore protect a physical process in the event of a cyber-attack on a CPS. The architecture is not specific to any one device, network, or process, so it can be adapted to many CPSes.

TAIGA functions by acting as a man-in-the middle between the embedded processor and physical system as shown in Figure 2. It provides a last line of defense to protect the physical process by monitoring the commands issued by the process controller. Instead of bolstering controller perimeter defenses, TAIGA adds a physical trusted layer of resiliency between the controller and the physical process. If the actions of the primary embedded controller deviate from acceptable predicted actions, TAIGA switches control over to an onboard backup controller, which is physically inaccessible to the network. Although the overall architecture of each TAIGA implementation is similar, the exact guards and backup actions will vary depending on the application. In systems where TAIGA has enough state sensor information to sustain the process without the primary controller, the autonomic backup controller may appear to act identically to the primary controller, and the process would continue uninhibited, though remote access capabilities would be lost. In most applications, however, it may be more reasonable to have TAIGA safely reduce or shut down the physical process, preventing physical damage. To ensure

system performance is not degraded by adding TAIGA, the architecture calls for interface controllers implemented in hardware [46].

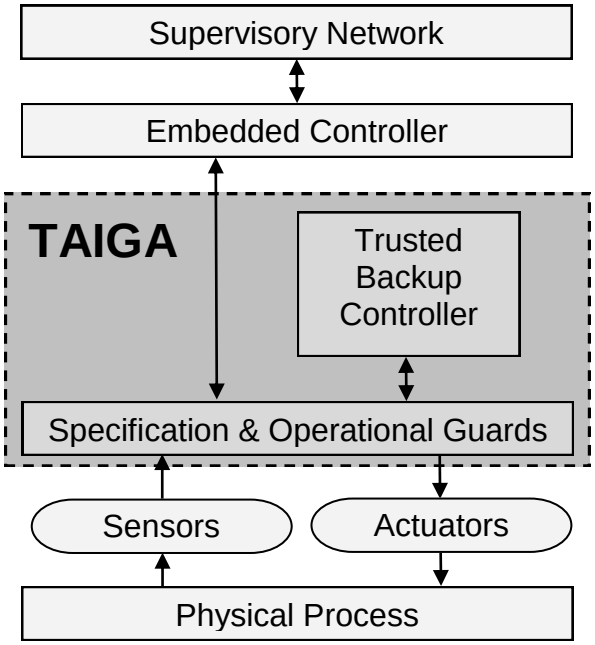

Figure 2: High-level overview of TAIGA

3.1 Detection and Mitigation of Attacks

To protect against attacks on the embedded controller, TAIGA must be able to identify and guard against the four attack outcome categories. To recognize abnormalities, TAIGA is provided with a variation of the primary embedded controller's control loop. It also stores the physical process's sensor values to update an internal model and predict future values of the process.

To detect possible attacks, TAIGA performs a series of logic checks as shown in Figure 3. Each TAIGA loop begins by checking the action of the primary controller by reading

any commands received from the primary controller. The TAIGA loop will run even if no controller commands are received under the implied action of *do nothing*.

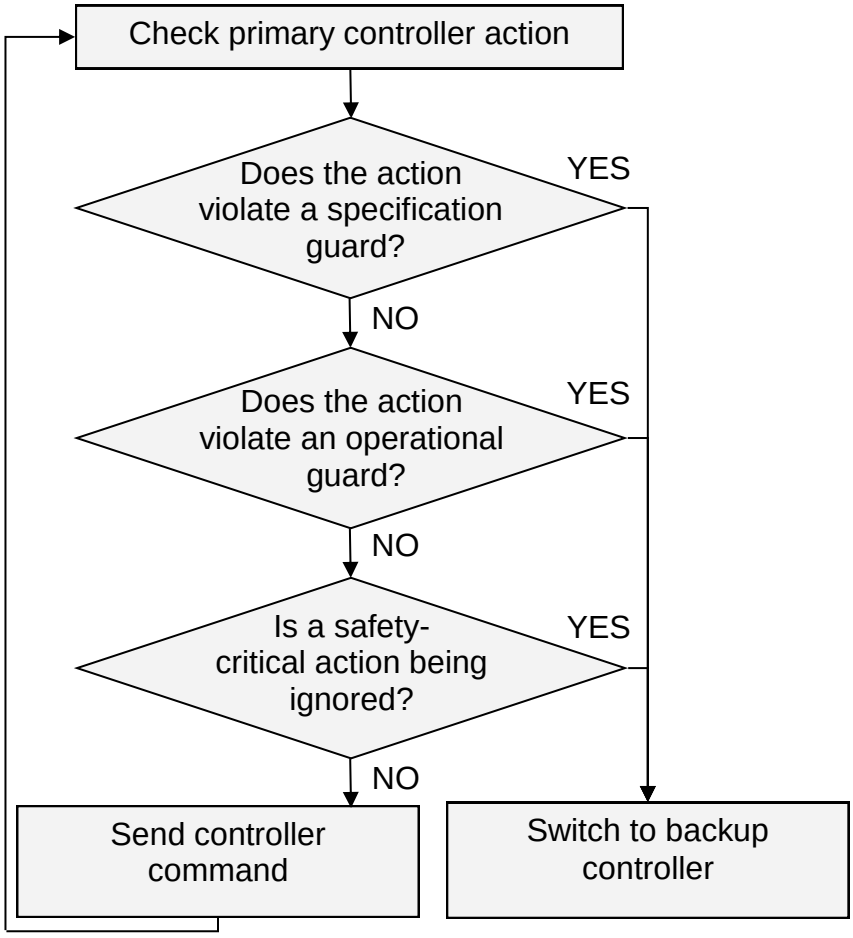

Figure 3: TAIGA guard logic check flow process

TAIGA first checks to see if the action violates a specification guard. Specification guards simply ensure that various parameters fall within specified ranges, such as the maximum speed of a motor, or pressure in a tank. If TAIGA detects that the command would violate a specification guard, it switches to its backup controller state. If the commands do not violate any specification guards, the command is then checked against operation guards. Operation guards are similar to specification guards, except that

they dynamically activate depending on the state of the physical process. For example, an autonomous automobile would reject commands to drive forward if it detected an object in front of it. If an operational guard is violated by the primary controller, TAIGA again switches to a backup control state. The final TAIGA check looks to see if the primary controller is neglecting any safety-critical actions. This is accomplished by checking the primary controller's action with predicted actions by TAIGA's version of the primary controller. This final check will trigger the backup controller if the primary controller is inactive or neglecting high priority commands.

In many systems, the desired parameter values of the various guards may slowly change over time. For example, the maximum safe speed of a turbine may decline slowly over time as components begin to wear, but may increase again after performing maintenance. To account for slight changes in parameters, TAIGA permits a tightly regulated parameter update process as described in [46]. The encrypted process only allows updates within certain preset value ranges, preventing an attacker from bypassing the system.

3.2 TAIGA's Trust Requirements Satisfiability

It is desirable to build a system to guard against the four possible attack outcomes on the embedded controller. It would be futile, however, if the guard system itself could be bypassed, or worse yet, the source of an attack. Any proposed guard system would need to designed, implemented, and validated such that the device can always be trusted. In order to place full trust in a guard system, it should satisfy five trust requirements (TRs) [10]:

TR1 The source code for the entire component is analyzed and compiled with trusted tools.

TR2 The component uses private computation, internal communication, and memory resources, and does not invoke external components as subfunctions.

TR3 All communication with other components is through bounded and isolated queues.

TR4 The component cannot be bypassed or disabled, and has a fixed repertoire of essential services, such as I/O or cryptography.

TR5 Critical functionalities of the component, such as rule checking logic, cannot be updated while active, and are only updatable through a maintenance port inaccessible to run-time networks.

Satisfaction of the TRs ensures that a device can be considered trustworthy throughout its lifetime. To ensure TAIGA can always be trusted, special precautions must be met during implementation to meet the five trust requirements. TR1 ensures the system is free of Trojan logic and hidden errors by analyzing hardware and software components in the pre-deployment stage. Relatively simple software can be evaluated using a series of testing simulations, formal verification techniques, and prototype implementations. To satisfy TR1, TAIGA's interface controller must be synthesized and formally analyzed using trusted tools. Because the guardian software functionality is relatively simple, it is able to be fully simulated and verified via static analysis.

While TR1 protects against errors arising internal to the system, TR2 and TR3 deal with protecting against errors arising from external systems. TR2 protects against unexpected component interactions that may occur if trusted and untrusted components share resources. To satisfy TR2, TAIGA must utilize true resource isolation and not share any hardware components with other systems. Achieving this is the most straightforward when TAIGA is implemented on a physically separate device.

Likewise, TR3 prohibits non-isolated communication between components which may allow untrusted components to introduce undesired behavior into the trusted components. In addition, isolated queues also safeguard the confidentiality of communications, which protects against API exploits such as Heartbleed. TAIGA satisfies TR3 because it uses dedicated, hardware-implemented queues for communication between devices.

TR4 and TR5 ensure that the component is not modified from its original design during run-time. They require that untrusted systems cannot workaround or turn off the security measures of the trusted device. The requirements can be met by requiring physical device access in order to utilize update ports. TR4 is satisfied when the logic guards are implemented and each command between the primary controller and process must be validated. Because the only means of communication between the processor and the controller is through TAIGA, the device cannot be bypassed. When the guards are isolated in hardware logic, they can only be updated by physical access to maintenance ports. The maintenance ports are physically separate from run-time networks, and thus cannot be updated remotely, satisfying TR5 [45].

CHAPTER 4

EXPERIMENTAL DESIGN AND RESULTS

To analyze TAIGA's capabilities, TAIGA was implemented on a demonstration hazardous cargo-carrying robot platform. The mobile robot setup simulates several industrial settings such as warehouses, nuclear power plants, and gas refineries [48] [49]. Similar robots are also used in high-risk emergency situations such as the ones used to investigate the damage at the Fukushima nuclear reactor [50]. These settings frequently use robots to perform tasks that are undesirable or unsafe for humans, and require a high amount of reliability since improper operation may result in significant physical harm.

This demonstration utilizes a small servo-controlled Jonny-Five humanoid robot sold by Lynxmotion as shown in Figure 4. The robot has differential drive treads and downward facing reflectivity sensors that were added to perform line-following. The robot tracks lines on the floor to move between cargo stations at the ends of the track as shown in Figure 5. A return ramp was added along the sides of the track so the cargo would automatically reset from the drop station to the retrieval station. This eliminated the need for a manual reset after each delivery, allowing the process to run indefinitely. The cargo was a mock-up flammable fuel tank fitted with a special handle for the robot to grab. The video file submitted with this thesis and listed in Figure 6 shows an overview of the experiment.

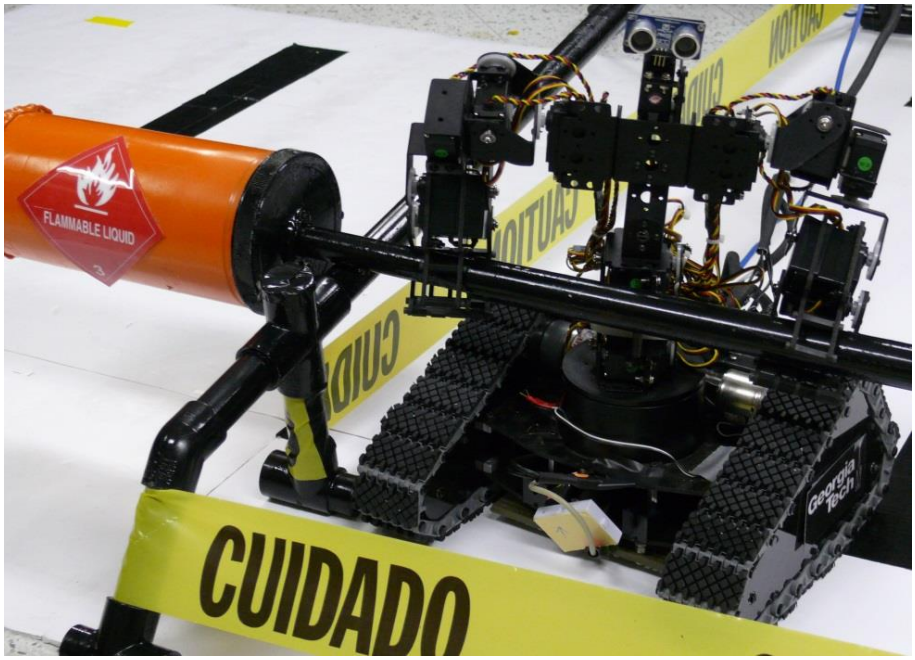

Figure 4: Demonstration robot and hazardous cargo

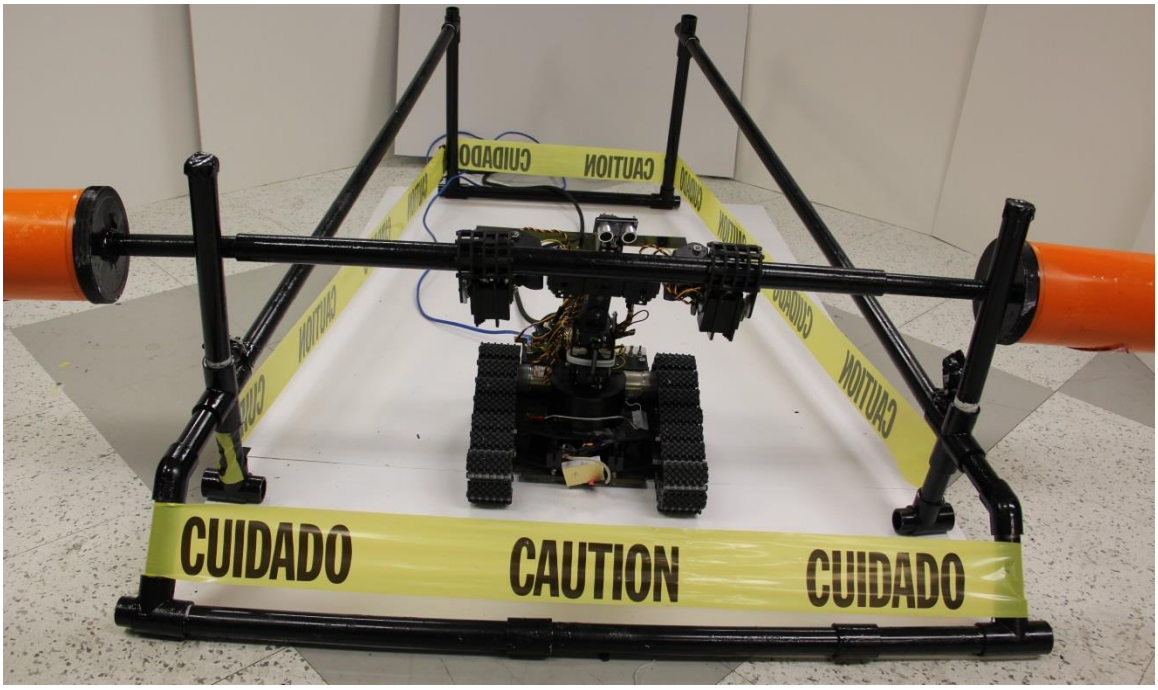

Figure 5: Experimental setup

Figure 6: Cargo demo overview video (student_kevinlyn_201507_fig6_Demo.wmv, 224K)

4.1 TAIGA Guard Design

During its operation, the robot moves between two control states as shown in Figure 7. The robot is in the *retrieving* state when it is driving forward with its hands open, ready to grab the cargo. Meanwhile, it makes slight velocity corrections as it performs forward line-following to stay on the painted path. When it detects reaching the cargo at the end of the path, the robot closes its claws and raises its arm to pick up the cargo, and transitions to the *delivering* state. The robot then holds onto its cargo as it traverses the path in reverse, until it again reaches the opposite station, where it releases the cargo and transitions back to the *retrieving* state.

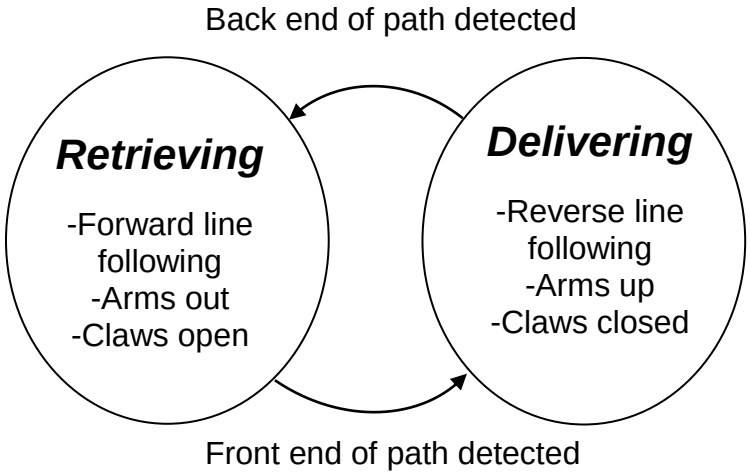

Figure 7: Robot states for cargo carrying operation

The specification guards for the implementation are shown in Table 2. The specification guards prevent the motors from damaging themselves by moving at unsafe

speeds. The operational guards, which vary depending on the state, ensure the robot is in the proper position to perform each task. Finally, the safety-critical check guards look for specific key corrective actions based on sensor data.

Table 2: Summary of implemented guards

Enforced	**Action**	**Value**
Specification Guards		
Always	Maximum Body Servo Speed	±90 deg/sec
Always	Maximum Tread Velocity	1 m/sec
Operational Guards		
Retrieving State	Arm Servo Position	Out
Retrieving State	Claw Servo Position	Open
Delivering State	Arm Servo Position	Up
Delivering State	Claw Servo Position	Closed
Safety-Critical Check Guards		
Front sensors off track	Forward line correction performed	
Back sensors off track	Reverse line correction performed	
Front end of the track reached	Arms raised, claws closed	
Rear end of the track reached	Arms lowered, claws opened	

Once an action fails a guard, the internal TAIGA back-up controller takes over the process. Because the control law is closed-loop and requires no information external to TAIGA, the backup controller provides the same functionality as the primary controller. While this experiment is ideally suited to demonstrate certain TAIGA components such as the back-up controller, it fails to utilize other functionalities, such as the prediction unit or vetting of controller updates.

4.2 TAIGA Implementation

The demonstration utilizes two Zynq-7000 All Programmable System-on-Chip (SoC) ZYBO development boards as shown in Figure 8 and Figure 9. The first board, outlined in red in Figure 9, shows the primary embedded controller. The primary controller runs a version of Ubuntu Linux on the board's ARM Cortex-A9 processor, and contains the robot's custom control algorithms written in C. Selected source code for key primary controller functions can be found in Appendix A. The controller communicates with the supervisory network via Ethernet. The second board, outlined in green, shows the hardware connections in the TAIGA implementation.

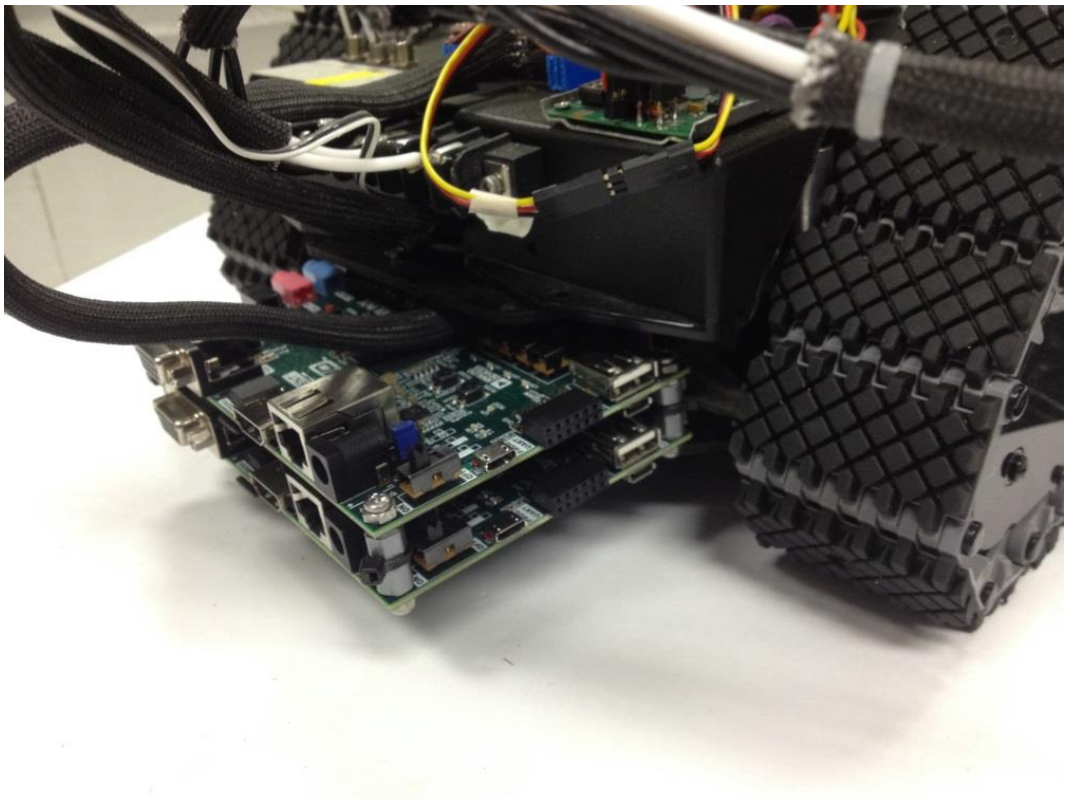

Figure 8: ZYBO boards used in the robot

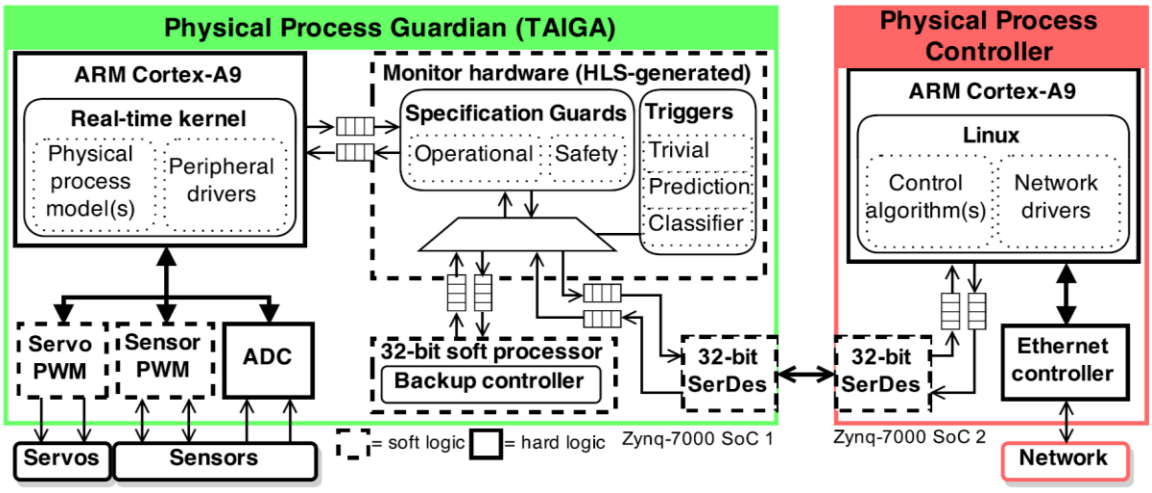

Figure 9: TAIGA implementation using two separate system-on-chip (SoC) devices

The two boards communicate only over a dual 32-bit bi-directional first-in-first-out (FIFO) serializer/deserializer (SerDes) interface. The communication interfaces on both boards were implemented in the programmable logic from C code using Xilinx's Vivado High-Level Synthesis (HLS) tool. Communication sent from the primary controller is then evaluated using the series of guard logic checks described previously in Figure 3. The logic checks were also implemented in the programmable logic using HLS. Once a command from the primary controller clears the logic checks, it is passed into a low-level controller running peripheral drivers to be executed by the robot's servos.

The low-level controller consists of another Arm Cortex-A9 processor running Free Real Time Operating System (FreeRTOS). In addition to controlling peripheral drivers, this processor stores sensor information, tracks the process, and provides a predictive model of future states and required actions in a similar fashion to the primary controller. The state information is readable by the logic guards, which allows the

dynamic operational guards to be enforced at the correct times. Meanwhile, the process backup controller is ready on a bare metal application, which is an application without an operating system, in a MicroBlaze soft processor. When commands from the primary controller fail to satisfy the logical guards, the backup controller is triggered and provided with the state information to generate corrective action. Once this has occurred, commands from the primary controller will be completely ignored, and the backup controller will command the process.

Table 3 shows the total configurable logic resources used on the Zynq-7000 development board to implement TAIGA, broken up by FPGA component. The table includes: flip-flops (FFs), look-up tables (LUTs), block RAM (BRAM), and 16-bit shift register look-up tables with clock enable (SRL16E). Overall, no more than 14% of any component was utilized in the implementation. This is important because it shows that TAIGA can be implemented without requiring excessive amounts of FPGA resources. Furthermore, it indicates that TAIGA may be built using off the shelf FPGA boards which may be fairly inexpensive relative to the process being protected. It should be noted that these values are specific to this implementation and will change depending on the complexity of other designs.

Table 3: TAIGA implementation resource usage

Component	**I/O**	**FF**	**LUT**	**BRAM**	**SRL16E**
Guardian Monitor	-	456	341	3	0
SerDes	-	706	346	2	1
Backup controller	-	1787	1833	2	132
Total Used	8	2949	2520	7	133
Total Available	100	35200	17600	60	6000
Percent Utilization	8%	8%	14%	12%	2%

Table 4 shows the sources of latency added by TAIGA. The largest source of latency comes from the SerDes link, which requires 70 clock cycles to transfer values from the FIFO to the SoC.

Table 4: TAIGA latency analysis

Component Latency		
Component	**Clock Frequency**	**Additional Clock Cycles**
	MHz	Min / Max
Monitor	150	2 / 7
32-bit SerDes	50 or 200	35 / 35
Timing Path Latency		
Path		**Worst Case Added Latency (ns)**
Linux to real-time kernel @ 50 MHz		1407
Linux to real-time kernel @ 200 MHz		38
Backup controller to real-time kernel		47

4.3 Code Verification and Testing

To ensure compliance with TR1, the code used in the TAIGA implementation was formally verified. Figure 10 shows a sample source code section defining a particular servo position operational guard. The range of acceptable values for the servo's position depends on the robot's state of either RETRIEVING or DELIVERING. The code contains ANSI/ISO C Specification Language (ACSL) annotations to perform the verification. The *verify_all_valid* and *verify_any_invalid* functions in line 17 were used to test all possible input values to the operational guards. The *assumes* statements in lines 10

and 14 provide the possible input ranges that should result in the function outputting the value denoted by the corresponding *ensures* statements.

```
1   typedef unsigned u8;
2   typedef unsigned u32;
3   enum States{RETRIEVING, DELIVERING};
4   #define SERVO_MAX 360
5   #define SERVO_MIN 0
6   #define validateThreshold(value, upper, lower)  ((value >= lower && value <= upper) ? 1 : 0)
7   // This function evaluates a servo position value against operational guards
8   /*@assigns \result;
9   behavior verify_all_valid:
10    assumes ((process_state == RETRIEVING)&& ((command <= SERVO_ MAX)&& (command >= 120))) ||
11          ((process_state == DELIVERING)&& ((command <= 90)&& (command >= SERVO_ MIN)));
12    ensures \result == 1;
13  behavior verify_any_invalid:
14    assumes ((process_state == RETRIEVING)&& ((command > SERVO_ MAX) || (command < 120))) ||
15          ((process_state == DELIVERING)&& ((command > 90) || (command < SERVO _MIN)));
16    ensures \result == 0;
17  disjoint behaviors verify_all_valid,verify_any_invalid;
18  complete behaviors verify_all_valid,verify_any_invalid; */
19  u8 servoPositionOperationalGuard(u32 command, enum States process_state){
20        u8 retval = 0;
21        if(process_state == RETRIEVING){
22                  retval = validateThreshold(command, SERVO _MAX, 120);}
23    else if(process_state == DELIVERING){
24                  retval = validateThreshold(command, 90, SERVO _MIN);}
25        return retval;}
```

Figure 10: Frama-C verification of operational guard C code

The verification of the operational guard proof was confirmed using the Alt-Ergo prover in Frama-C's Jessie plugin [51]. The analysis functions *disjoint behaviors* and *complete behaviors* in lines 17 and 18 designate that the evaluation set of behaviors as complete and disjoint. When the code passed verification, this meant that all possible input values achieved the desired result [10].

While efforts were made to formally verify as much of the source code as possible, certain areas of the implementation could not be formally tested. For example, the backup controller code runs on an Arm Cortex-A9 running FreeRTOS. While the backup

controller code could be verified, the entire FreeRTOS package could not. This could be solved by using a simpler RTOS package that can be verified, or developing a custom replacement RTOS using only trusted tools. Another alternative would be to use a RTOS version which has been designed, tested, and certified for critical CPS use, such as SafeRTOS [52].

4.4 Simulated Attacks & Results

To test the effectiveness of TAIGA, cyber-attacks were launched resulting in each of the four potential outcomes, each with and without TAIGA in place. Each setup was tested ten times and run until the process was physically damaged, or for a maximum of ten minutes. The process was considered damaged when either the robot or cargo collided with another object, such as a wall or cargo station piece, as shown in Figure 11.

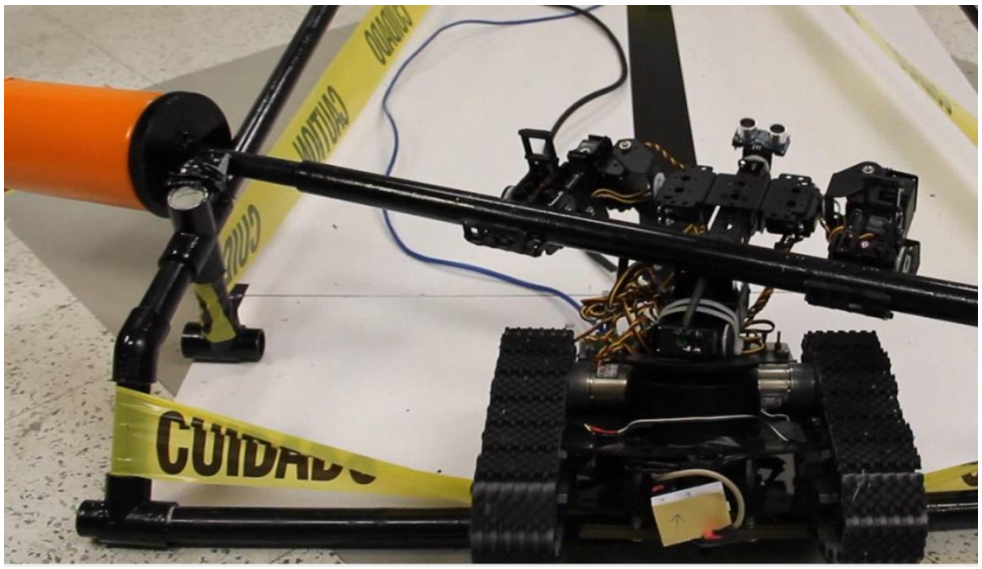

Figure 11: Robot dropping the cargo as it breaks through a barrier

The Controller Non-Functional effect was simulated by sending a malicious update to the processor. The update contained an infinite *while* loop in the control

algorithm, so the control code would stop sending commands shortly after beginning. Without TAIGA implemented, the robot's motors maintained the same value when the controller locked-up, and continued to drive forward off the track, resulting in a collision with the mock explosive fuel tank. With TAIGA, however, the safety-critical command guards detected that the robot had reached the end of the track, but no command to reverse was received. The back-up controller took command and restored the process to normal operation.

To test the resilience to the Controller Incorrect effect, a Trojan script was planted in the processor. Randomly activated once in every thirty second interval, the Trojan would modify the values of random variables in the control loop. Without TAIGA, the robot began exhibiting jerky motions or performing actions at the incorrect time. Over time, the random interruptions created the correct conditions for the robot to collide with or drop the hazardous fuel tank. With TAIGA, however, the guards quickly identified incorrect behavior, and engaged the backup controller.

To determine the resilience to the Controller DoS effect, a program on the supervisory network sent a continuous stream of commands to drive forward at various speeds. Without TAIGA, the time processing the excess commands delayed sensor polling, making the robot unable to react to ends of the track. As a result, the robot quickly drove off the track and into the hazardous cargo. With TAIGA, however, the guards identified that the sensors were not being polled as frequently as required, and as a result would invoke the back-up controller.

For the final test, an application program interface (API) was provided to an intelligent hacker to simulate the Malicious Controller effect. The hacker had the ability to monitor sensor readings, issue any level of command, and change any available variable in real-time. With full access to the system, the hacker easily commanded the robot to throw the explosive fuel tank into a wall. With the addition of TAIGA, however, the hacker's actions quickly triggered the backup controller. Because TAIGA only acts as a monitor during normal operation, the hacker was unaware of TAIGA's presence until after it had taken control of the process. Even then, the hacker was unable to modify or disable TAIGA.

Recorded results are summarized in Table 5. The first set of data shows the average time between the start of a cyber-attack, and when the process damaged itself without TAIGA. For most attacks, physical damage could be accomplished in seconds, or at most, minutes. The table also shows a second set of data with TAIGA, which lists the average time between the start of the cyber-attack, and when the back-up controller takes over the process. In every case, TAIGA was able to detect the attack, and the backup controller was able to take control. While attacks were able to cause physical harm without TAIGA, they were unable to disrupt the process with TAIGA. When read together, the results indicate that the backup controller has enough time to restore stability to the process once an attack has started.

Table 5: Experimental results with and without TAIGA

	Without TAIGA		With TAIGA	
Cyber-attack and Embedded Controller Outcome	**Outcome**	**Seconds until process damage**	**Outcome**	**Seconds until backup controller takeover**
Controller Non-Functional	Physical damage	5	Process uninterrupted	3
Controller Incorrect	Physical damage	225	Process uninterrupted	57
Controller DoS	Physical damage	6	Process uninterrupted	4
Malicious Controller	Physical damage	3	Process uninterrupted	1

CHAPTER 5

TAIGA GENERALIZATIONS

The results presented in the previous chapter illustrate an example of TAIGA's capability to maintain a process without interruption even during a cyber-attack. But the exact implementation of TAIGA varies depending on the process application. Certain aspects, such as the backup controller and specification guards, are designed using application-specific control algorithms and logic [45]. Furthermore, the availability of other features, such as the prediction unit, depends on the system configuration [10]. These system-specific conditions raise an important question, *how can TAIGA be generalized to help protect any CPS*? Specifically, what system-level criteria must be met to receive the various protection benefits offered by TAIGA? This chapter will aim to answer this question. The first section will begin by establishing some basic control-theory terms and definitions used in later analysis. The second section will then provide a system-level categorization of TAIGA's possible benefits. After that, various modifications will be proposed to the experimental setup described in the previous chapter, and its impact on TAIGA's benefits will be discussed. Finally, this chapter will present a set of system-level criteria that determine which benefits TAIGA can provide to any given system.

5.1 Control Theory Definitions

Before beginning our analysis of TAIGA's benefits, we define some common control-theory terminology that will be used in later sections. We start by defining terms

used in conjunction with the state-space model representation of a system, commonly used in control design. The state-space model consists of a *p* x *1* input vector $\boldsymbol{u}(t)$, an *n* x *1* internal state variables $\boldsymbol{x}(t)$, and a *q* x *1* external output variables $\boldsymbol{y}(t)$. The variables are related through the *n* x *n* system matrix $\boldsymbol{A}(t)$, *n* x *p* input matrix $\boldsymbol{B}(t)$, *q* x *n* output matrix $\boldsymbol{C}(t)$, and *q* x *p* feedthrough matrix $\boldsymbol{D}(t)$, as shown in equations (1) and (2) [53] . For a linear and time invariant (LTI) system, the system matrices are not time dependent and can be rewritten as **A**, **B**, **C** and **D**.

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}\boldsymbol{u}(t) \quad (1)$$
$$\boldsymbol{y}(t) = \boldsymbol{C}\boldsymbol{x}(t) + \boldsymbol{D}\boldsymbol{u}(t) \quad (2)$$

Control system engineers define a system to be *controllable* if the system's actuators are capable of exercising control over all system states. In other words, a system is controllable iff $\exists\ \boldsymbol{u}(t)$ *s.t.* $\boldsymbol{x}(t_o)$ can be driven to $\boldsymbol{x}(t_f)$, $\forall\ \boldsymbol{x}(t_o)$. A simple test for controllability can be performed by checking that the rank of the controllability matrix $\boldsymbol{R}$, shown in equation (3), is equal to *n*, the number of system states [54]. A similar but weaker criterion is required for a system to be *stabilizable*, which allows for uncontrolled states if and only if those states have naturally stable dynamics. It should be noted that both controllability and stabilizability depend only on the relationship between the system dynamic and input dynamic matrices $\boldsymbol{A}$ and $\boldsymbol{B}$, and do not account for the system's ability to observe the output with its sensors.

$$\boldsymbol{R} = [\ \boldsymbol{B}\ \boldsymbol{A}\boldsymbol{B}\ \boldsymbol{A}^2\boldsymbol{B}\ \ldots\ \boldsymbol{A}^{n-1}\boldsymbol{B}\ ] \quad (3)$$

If the system's states can be determined from the output sensors, the system is said to be *observable*. This occurs if all system states $\boldsymbol{x}(t)$ impact $\boldsymbol{y}(t)$ in such a way that $\boldsymbol{x}(t)$ can be determined from $\boldsymbol{y}(t)$. Mathematically, this occurs when the observability

matrix $\boldsymbol{Q}$ shown in equation (4) has full rank [55]. If a system is not observable, it may be *detectable* if the unobservable states asymptotically decay. It should again be noted that observability and detectability only depend on the system dynamics and output dynamics **A** and **C**, and are unaffected by the ability of the system to react to observed states.

$$\boldsymbol{Q} = \begin{bmatrix} \boldsymbol{C} \\ \boldsymbol{CA} \\ \boldsymbol{CA}^2 \\ \vdots \\ \boldsymbol{CA}^{n-1} \end{bmatrix} \qquad (4)$$

5.1.1 System with Feedback

Most control systems are further related through the feedback loop shown in Figure 12. In the figure, the reference signal ***r(t)*** represents the desired value of the output states. The system attempts to drive ***y(t)*** to ***r(t)*** using the feedback loop relationship provided by the feedback gain matrix ***K***. As a result of the feedback relationship, ***u(t)*** can be computed using equation (5).

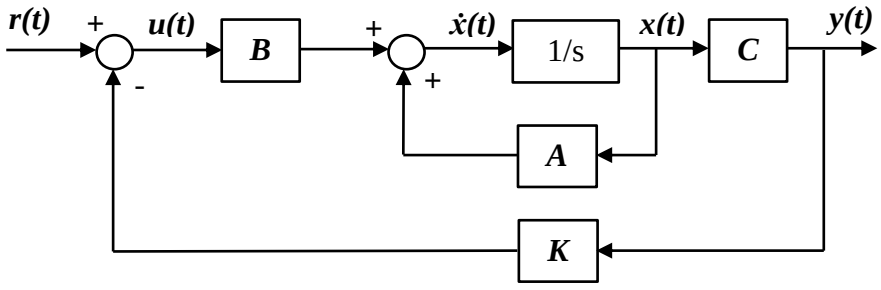

Figure 12: Typical state-space feedback loop

$$\boldsymbol{u}(t) = -\boldsymbol{K}\boldsymbol{y}(t) + \boldsymbol{r}(t) \qquad (5)$$

It should be noted that $\boldsymbol{r(t)}$ is generated by the high-level control law for the system. For remotely operated systems, such as a remotely controlled aircraft, $\boldsymbol{r(t)}$ is determined by the aircraft pilot's control inputs. In truly autonomous systems, such as a manipulator at a factory performing repetitive tasks, $\boldsymbol{r(t)}$ may be preprogrammed ahead of time.

Although feedback is focused on driving $\boldsymbol{y(t)}$ to $\boldsymbol{r(t)}$, the safety status of the process is often dependent on $\boldsymbol{x(t)}$. In an ideal case, the system would know the precise values of $\boldsymbol{x(t)}$ even under open loop conditions, unfortunately, there are very few systems that can operate in this fashion [56]. It is therefore critical that the system can both monitor and control $\boldsymbol{x(t)}$, however, equation (5) does not include an $\boldsymbol{x(t)}$ term. Recall, however, that $\boldsymbol{y(t)}$ and $\boldsymbol{x(t)}$ are related through equation (2), and that the $\boldsymbol{x(t)}$ terms can be observed and controlled if the system is completely observable and controllable.

5.2 Categorization of TAIGA's Benefits

With the control-theory terms defined, we seek to relate them to TAIGA's possible benefits. It is important to note that a benefit provides an observable effect to the system, while a feature is a capability of TAIGA used to produce a benefit. Although TAIGA includes several noteworthy features, its key benefits include process maintainability and safety. But how is "safety" defined? For each point in time, we can define a safety set $\boldsymbol{S(t)}$, where as long as the system states, $\boldsymbol{x(t)}$, lie within $\mathbf{S}(t)$, the system is safe. Reworded, a process is *safe* if $\boldsymbol{x(t) \in S(t)}$ [28]. We note that $\boldsymbol{S(t)}$ may be time-varying, representing that the requirements of the system may change dynamically over time. For example, it may be acceptable for an autonomous automobile to travel forward at a given velocity, but once it detects an object ahead, it must either stop or change

directions. While $\boldsymbol{S(t)}$ usually does not often appear explicitly in control design, designers often account for restrictions placed by $\boldsymbol{S(t)}$ in the form of constraints.

Although a system may be safe, that does not necessarily mean it is acting as desired. For example, the autonomous automobile may be perfectly safe in its garage, but would be rather useless if it only stayed there. When operating normally, the system should attempt to follow the desired feasible process reference signal $\boldsymbol{r^*(t)}$, though because no system is truly ideal, it cannot perfectly follow $\boldsymbol{r^*(t)}$. As a result, let $\boldsymbol{x^*(t)}$ and $\boldsymbol{y^*(t)}$ represent the realized results from the input $\boldsymbol{u^*(t)}$, which is attempting to follow $\boldsymbol{r^*(t)}$. It should be noted that physical errors and variations may result in different actual values for $\boldsymbol{u^*(t)}$, $\boldsymbol{x^*(t)}$, and $\boldsymbol{y^*(t)}$ for the same $\boldsymbol{r^*(t)}$. This is expected because $\boldsymbol{u^*(t)}$, $\boldsymbol{x^*(t)}$ and $\boldsymbol{y^*(t)}$ only represent the best possible physical realization of $\boldsymbol{u(t)}$, $\boldsymbol{x(t)}$ and $\boldsymbol{y(t)}$ in the face the system imperfections at the time of realization. We can therefore say the *process is maintained* if the realized states $\mathbf{x}\boldsymbol{(t)} = \boldsymbol{x^*(t)}$. It is important to note the feasibility restriction placed on $\boldsymbol{r^*(t)}$. This physical restriction indicates that $\boldsymbol{r^*(t)}$ must be met without actuator saturation [57].

As with any fault mitigation system, TAIGA's responsiveness is determined by its ability to detect when the process is not maintained or is unsafe. If TAIGA's specification guards adequately detect faults, we say TAIGA offers *automatic* protection. Furthermore, if TAIGA is equipped with a trusted warning device such as audible or visual alarms, TAIGA offers *alarmability*. If TAIGA includes a trusted manual input device, such as a trusted emergency stop (e-stop) switch, we say TAIGA includes *manual* process interfaces.

By combining and rearranging the previously discussed terms, we define three possible TAIGA process benefits and three possible safety benefits:

Possible Process Benefits:

- ***Automatic Process Maintainability*** **–** A cyber-attack will automatically trigger the backup controller which maintains the process as if no attack occurred, such that the realized $\boldsymbol{r(t)} = \boldsymbol{r^*(t)} \; \forall \; t$.
- ***Manual Process Maintainability*** **–**TAIGA features a trusted HMI where an operator is able to provide $\boldsymbol{r^*(t)}$ and therefor $\boldsymbol{u^*(t)}$ to maintain the process.
- ***Process Alarmability*** **–** TAIGA is able to automatically raise a trusted local alarm when process is predicted to deviate outside expected process limits such that $|\boldsymbol{x(t)} - \boldsymbol{x^*(t)}|$ < the maximum expected process error.

Possible Safety Benefits:

- ***Automatic Safety Maintainability*** **-** A cyber-attack will automatically trigger the backup controller which can ensure system safety such that $\boldsymbol{x(t)} \in \boldsymbol{S(t)} \; \forall \; t$.
- ***Manual Safety Maintainability*** **–** The backup controller includes a manual trigger to switchover, and once activated can ensure system safety such that $\boldsymbol{x(t)} \in \boldsymbol{S(t)} \; \forall \; t$ if $\boldsymbol{x(t)}$ is recoverable (can be returned to $\boldsymbol{S(t)}$) at the time of switchover [28].
- ***Safety Alarmability*** **–** TAIGA is able to automatically raise a trusted local alarm when the system is predicated to be unsafe such that $\boldsymbol{x(t)} \Subset \boldsymbol{S(t)}$.

Several of the terms listed above contrast from typical terminology used in TAIGA literature. This is because most previously discussed systems, such as the

experiment in Chapter 4, are ideally suited for TAIGA and therefore meet the criterion for highly-desirable benefits such as *automatic process maintainability*. In such a context, the discussion focuses on the single maximum achievable benefit, and ignores other benefits that may be more applicable to other systems.

For example, *Automatic process maintainability* is not possible on systems that are not fully autonomous, yet TAIGA can provide other benefits to non-autonomous systems. For example, it may be desirable for an industrial plant to implement TAIGA only as a trusted cyber-attack alarm which notifies plant personal who can stabilize the system using mechanical interlocks [58]. Similarly, in the event of an unexpected emergency due to a cyber-attack, a system operator may wish to manually activate TAIGA to reduce process load without shutting off power to the entire system. For example, during a cyber-attack, the emergency stop button on a full-sized humanoid mobile robot could activate TAIGA's trusted backup controller to freeze the robot's actions while still preventing the robot from falling over [59].

But the question remains: what are the system requirements to achieve the certain benefits listed above? In the next section, we modify the experiment in Chapter 4 to see how the system's eligible benefits change depending on the system configuration.

5.3 System Configuration Affects on Eligible Benefits

Throughout its development, the refined experiment presented in Chapter 4 underwent several major revisions and variations. One such version involved a flag signaling scenario, where the robot raised a specific flag upon user request, while rejecting malicious commands to raise an incorrect flag in the other opposite hand. A

second setup involved a similar hazardous-cargo transportation setup, but called for three cargo stations, two colors of tape, and nineteen control states. Both setups are described in greater detail in Appendix B. Furthermore, others pursued additional implementations of TAIGA that have been evaluated using a cognitive radio [44], simulated aircraft pitch controller [21], simulated PID controller [45], and inverted pendulum [60] [61]. During the development of these experiments, it was realized that the benefits of TAIGA change depending on how the scenario is built. Specifically, the benefits change based on the location and capabilities of the sensors, actuators, control laws, and trusted HMIs. We can therefore generate interesting thought experiments by adding, removing, or rearranging these components in the experimental setup previously mentioned in Chapter 4.

5.3.1 Robot Model

In order to guarantee any level process control, we assume the sensors and actuators that are connected only to TAIGA and are fairly simple devices without significant capacity to execute cyber-attacks of their own [62]. Additional research is required before interconnected or more complex embedded devices can be added to TAIGA. Although a cyber-attack is not expected to originate from these devices, physical device failure is still possible. There are many studies on designing sufficient system redundancies to guard against physical failures [63] [64], but since they do not affect the cyber-defenses provided by TAIGA, they are not discussed here. We also note that, although TAIGA has a prediction unit, the model still relies on sensor feedback to bound the prediction unit drift error. While the prediction unit has many important uses, it cannot be used as a trusted sensor substitute.

In addition to assuming a cyber-attack will not originate from the sensors and actuators interfacing only with TAIGA, we also assume attackers do not have physical device access. If attackers had device access, they would have more direct methods to damage the process, and are therefore a different threat than a pure cyber-attack. Physical access also includes access to trusted local switches and interfaces connected only to TAIGA.

The experiment in Chapter 4 utilized sixteen decoupled motors, resulting in a thirty-two variable state vector defined by the motor positions and their corresponding velocities. The resulting state-space model would have sparse A, B and C matrices, and a zero D matrix. Despite the model's complexity, the states are grouped into sixteen independent pose state pairs. Because of the independent nature of the states, changing a sensor or actuator will affect at most one pair of pose states, so the remaining unaffected states can therefore be ignored in the discussion. For simplicity, we include only the servos states of significance: the robot's forward translation pose, the arm servo's angular pose, and the claw servo's angular pose. The state-space representation of the robot is shown in equation (6).

Let x_1 be the robot's forward translational position, x_2 be the arm servo's angular position, and x_3 the claw servo's angular position. The related velocities are $x_4 = \dot{x}_1$, $x_5 = \dot{x}_2$, and $x_6 = \dot{x}_3$. The motors' angular acceleration, $\dot{x}_4$, $\dot{x}_5$, and $\dot{x}_6$, are controlled by the command inputs u_1, u_2, and u_3 respectively. Because the positions are observed by the sensors, let $y_1 = c_{11}x_1$, $y_2 = c_{22}x_2$, and $y_3 = c_{33}x_3$, where c_{11}, c_{22}, and c_{33} are non-zero scaling constants.

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\\ \dot{x}_3\\ \dot{x}_4\\ \dot{x}_5\\ \dot{x}_6\end{bmatrix} = \begin{bmatrix}0 & 0 & 0 & a_{14} & 0 & 0\\ 0 & 0 & 0 & 0 & a_{25} & 0\\ 0 & 0 & 0 & 0 & 0 & a_{36}\\ 0 & 0 & 0 & 0 & 0 & 0\\ 0 & a_{52} & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 0\end{bmatrix}\begin{bmatrix}x_1\\ x_2\\ x_3\\ x_4\\ x_5\\ x_6\end{bmatrix} + \begin{bmatrix}0 & 0 & 0\\ 0 & 0 & 0\\ 0 & 0 & 0\\ b_{41} & 0 & 0\\ 0 & b_{52} & 0\\ 0 & 0 & b_{63}\end{bmatrix}\begin{bmatrix}u_1\\ u_2\\ u_3\end{bmatrix} \qquad (6)$$

$$\begin{bmatrix}y_1\\ y_2\\ y_3\end{bmatrix} = \begin{bmatrix}c_{11} & 0 & 0 & 0 & 0 & 0\\ 0 & c_{22} & 0 & 0 & 0 & 0\\ 0 & 0 & c_{33} & 0 & 0 & 0\end{bmatrix}\begin{bmatrix}x_1\\ x_2\\ x_3\\ x_4\\ x_5\\ x_6\end{bmatrix}$$

The model assumes the motors are sufficiently geared such that the robot will not move on the horizontal surface without actively being driven, and the claw will not passively open or close. Nonlinear effects such as friction as well as linear effects such as spring constants and damping are relatively small and are ignored. The arms, however, experience a significant gravitational force when raised, and therefore a_{52} signifies a linear approximation of the gravitational force returning the arms to their lowered resting position.

The configuration presented in Figure 13 shows a detailed hardware setup of the experiment performed in Chapter 4. Note that all sensors and actuators are connected directly to TAIGA. In addition, TAIGA can select between two input signals: an external $\boldsymbol{u}_{\_controller}$, which originates from the untrusted primary controller, or an internal $\boldsymbol{u}^*$, which originates from TAIGA's own trusted model of the process controller. Because the process in the experiment was entirely autonomous, TAIGA has been preprogrammed with an internal $\boldsymbol{r}^*_{\_TAIGA}$ and set of control laws which should be nearly identical to $\boldsymbol{r}^*$

used in the primary controller's control laws. With knowledge of ***r**** and the control laws, TAIGA can internally determine ***u***.*

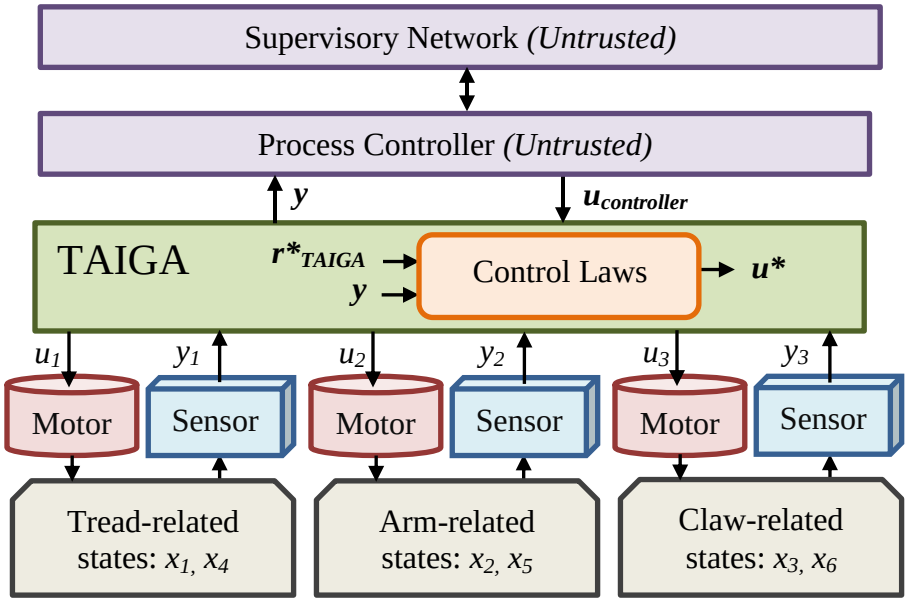

Figure 13: Original system configuration

Performing the observability and controllability rank test on equation (6), the system is both completely observable and completely controllable. Furthermore, TAIGA is preprogramed with trusted internal $\boldsymbol{r}^*_{\_TAIGA,}$ which it can use to generate ***u***(t)*. This setup therefore receives accurate state information, contains sufficient process knowledge to internally determine ***u***(t),* and has sufficient trusted actuators to perform the desired action. In this case, TAIGA can both detect and automatically correct for excessive deviations in ***u**(t)* from ***u***(t)*. We can therefore say that the setup mentioned in Chapter 4 has *Automatic Safety Maintainability* and *Automatic Process Maintainability*. The experimental results confirm this conclusion.

It should be noted that although the setup can automatically maintain and protect itself, it lacks the required HMI components to satisfy either the *Manual Process Maintainability* or the *Manual Safety Maintainability* requirements. This implies that only a violation of the specification guards can trigger the backup controller, and a nearby human operator would have no direct ability to request an emergency backup controller switch. In addition, because a well-designed attack may involve a stealth component, the primary controller may be modified to ignore any alarms generated by TAIGA. Without additional trusted alarm components, the setup fails to satisfy either the *Process Alarmability* or *Safety Alarmability* requirements. As a result, a nearby operator may not realize TAIGA has taken process control until they notice TAIGA rejecting commands from the primary controller.

5.3.2 Sensor & E-Stop Placement

Consider the proposed modification shown in Figure 14. The setup is similar to Figure 13; however the locally connected reflectivity sensor which previously provided the output reading y_1 was replaced with an externally connected acoustic ping sensor fixed to one end of the track. While the same state information can be read for both sensors, the acoustic sensor is connected to the process controller through the supervisory network, and does communicate directly through TAIGA. Although this setup for the robot experiment seems unnecessarily complex, the setup closely resembles many CPSes that cover large geographic areas or require remotely placed sensor nodes. For example, some modern agricultural irrigation systems use feedback from remote sensors that measure evapotranspiration and other environmental field parameters [65]. Because the network or process controller may contain malware affecting the sensor values, the

external sensor is untrusted. Although control-theory research has identified methods to deal with sensor delays, packet loss, or data-injection [66] [67], they rely on either using redundant sensors, or on some form of correct information eventually arriving at the controller. Because an attack on the sensor, controller, or network may prevent any accurate information from ever reaching the controller, the sensor must be ignored entirely from within TAIGA. This results in $c_{11} = 0$, with the modified output now shown in equation (7). A quick observability calculation reveals that this system is no longer observable or detectable within TAIGA as x_1 was only observable through y_1, and x_1 is not a naturally bounded state. As a result, TAIGA cannot provide any significant benefits to this system since it does not have the information required to determine an accurate ***y(t)*** or ***x(t)***.

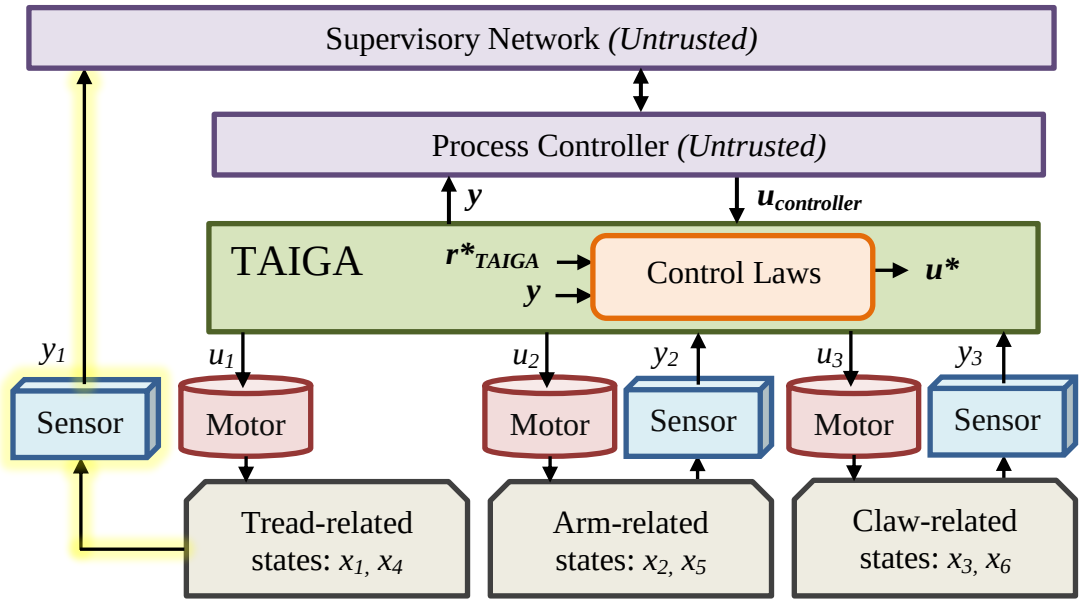

Figure 14: Configuration unable to provide any benefits due to untrusted sensor *y1*

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} \mathbf{0} & 0 & 0 & 0 & 0 & 0 \\ 0 & c_{22} & 0 & 0 & 0 & 0 \\ 0 & 0 & c_{33} & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} \quad (7)$$

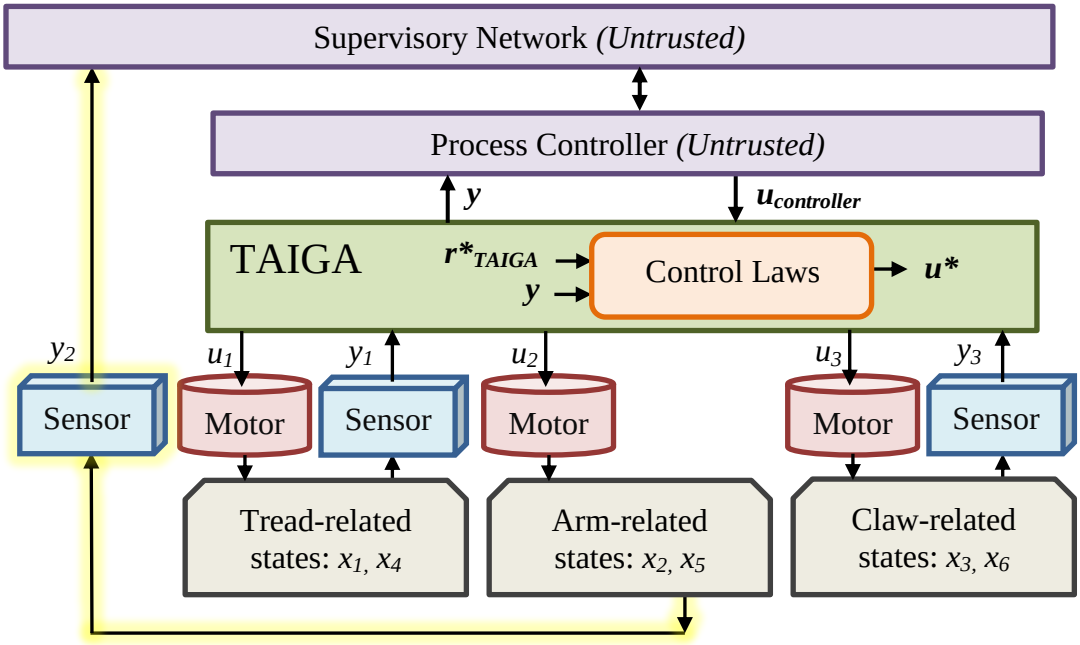

Figure 15: Configuration able to provide limited benefits due to untrusted sensor *y2*

The next configuration is presented in Figure 15. Instead of a position encoder internal to TAIGA, the system now receives arm positioning information using a set of active inferred reflection sensors fixed along track. At first glance, this setup appears nearly identical to the previous setup in Figure 14. With the external sensor untrusted, the new setup's ***C*** matrix is shown in equation (8). Upon analysis, however, we note that the unobservable states x_2 and x_5 will decay to a natural stable resting position if no input is received. This means that the system is detectable within TAIGA. If the process reference signal ***r***(t)* calls for non-zero inputs ***u***(t)* which affect the unobservable states, however,

TAIGA cannot observe if x_2 and x_5 are within the safety set $\boldsymbol{S}$. Without knowledge of the safety status of x_2 and x_5, TAIGA cannot offer any benefit to this process under the above conditions.

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} c_{11} & 0 & 0 & 0 & 0 & 0 \\ 0 & \mathbf{0} & 0 & 0 & 0 & 0 \\ 0 & 0 & c_{33} & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} \quad (8)$$

But if the process signal $\boldsymbol{r^*(t)}$ calls for the inputs impacting the unobserved states $\boldsymbol{u}_{unobservable}\boldsymbol{^*(t)}=0 \ \forall \ t$, then the unobservable system will stabilize naturally over time. If the unobserved state naturally stays within $\boldsymbol{S}(t) \ \forall \ t$ and $\boldsymbol{u}_{unobservable}(t)=0 \ \forall \ t$, then those states may be considered always safe. Because the unobservable states are safe, and the observable state can be monitored and corrected, the system is eligible for *Automatic Safety Maintainability***.**

The setup is again modified as shown in Figure 16. The setup is identical to the previous setup, except a local simple external physical emergency stop switch has been added which invokes the TAIGA backup controller. Such switches are especially common on autonomous vehicles and mobile robots where simply disconnecting device power is insufficient to bring the device to a safe state [59] [68]. Because TAIGA has direct access to the simple analog switch, and the switch is not networked with any other device, it is assumed that a cyber-attack will not originate from the switch. The switch does not change equation (8), and our previous analysis that the system is detectable still holds. The addition of the switch, however, means that a local operator may identify undesirable or unsafe system behavior, and activate the e-stop switch. Since the system is

detectable, by cutting off any inputs to the unobservable states, those states will self-stabilize to their natural stable value. If this stable value lies within $\boldsymbol{S(t)}\ \forall\ t$, then TAIGA can restore system safety and move all values within $\boldsymbol{S(t)}$. It can therefore be said that TAIGA offers *Manual Safety Maintainability*. TAIGA may still offer *Automatic Safety Maintainability* if it satisfies the $\boldsymbol{u}_{unobservable}\boldsymbol{*(t)}=0\ \forall\ t$ discussed in the previous setup.

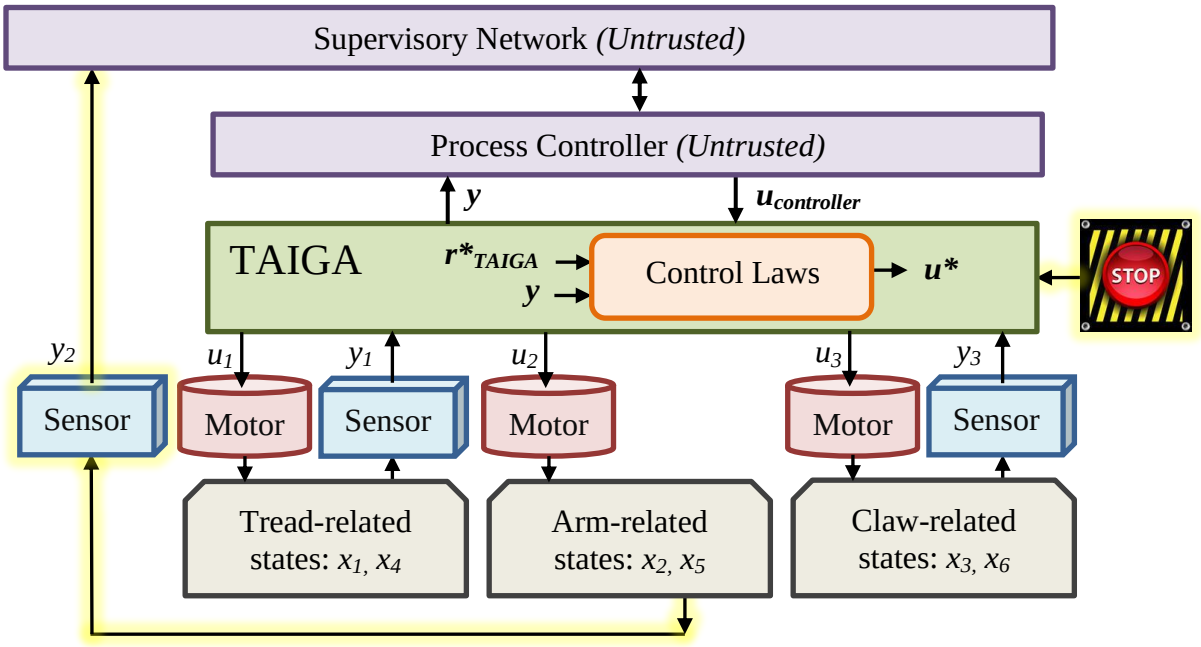

Figure 16: Configuration able to provide manual benefits due to a switchover trigger

5.3.3 Actuator Placement

Figure 17 presents an actuator modification to the originally discussed system. The figure shows the tread motors, which originally connected to TAIGA, instead connected to the process controller via the supervisory network. While this setup again appears impractical for the robot application, the setup models many CPSes which require actuators across a large area. For example, a combustion turbine generator at a

power station relies on many auxiliary systems, such as cooling water and seal oil, whose valves and pumps may be physically distant from the actual generator [69]. Because the motor lies outside of TAIGA, it is untrusted and completely vulnerable to cyber-attacks. Because TAIGA can no longer control the motor through a trusted connection, b_{41} is replaced with a zero as shown in equation (9). Furthermore, because the actuator is vulnerable, an attacker may take control and provide malicious inputs, indicated by the malicious control vector ***m(t)***. The malicious inputs are affected by the malicious input dynamics represented by the matrix ***N***. Because the attacker can disrupt or destroy the process, TAIGA cannot provide any benefits in this configuration.

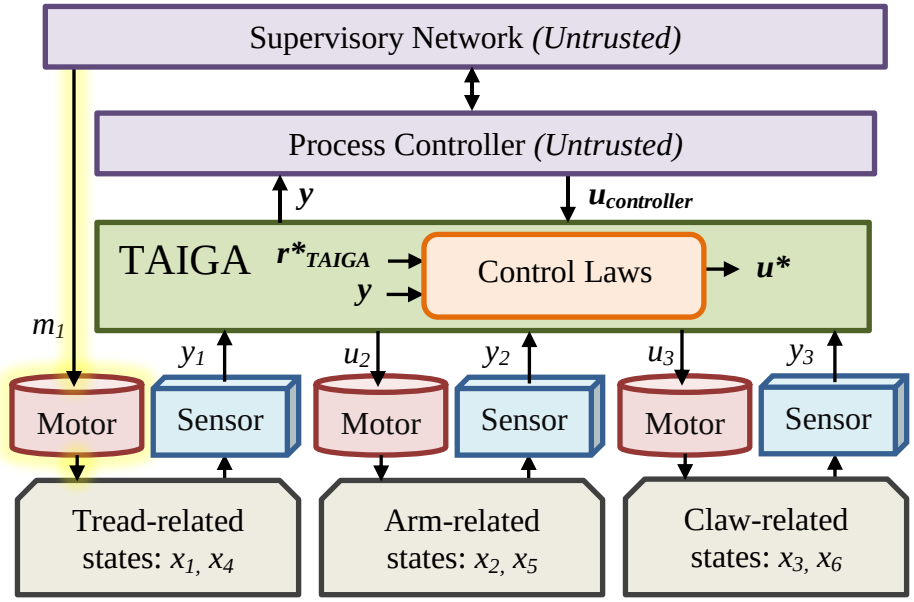

Figure 17: Configuration unable to provide benefits due to untrusted actuator u_1

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\\ \dot{x}_3\\ \dot{x}_4\\ \dot{x}_5\\ \dot{x}_6\end{bmatrix} = \begin{bmatrix}0 & 0 & 0 & a_{14} & 0 & 0\\ 0 & 0 & 0 & 0 & a_{25} & 0\\ 0 & 0 & 0 & 0 & 0 & a_{36}\\ 0 & 0 & 0 & 0 & 0 & 0\\ 0 & a_{52} & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 0\end{bmatrix}\begin{bmatrix}x_1\\ x_2\\ x_3\\ x_4\\ x_5\\ x_6\end{bmatrix} + \begin{bmatrix}0 & 0 & 0\\ 0 & 0 & 0\\ 0 & 0 & 0\\ \mathbf{0} & 0 & 0\\ 0 & b_{52} & 0\\ 0 & 0 & b_{63}\end{bmatrix}\begin{bmatrix}u_1\\ u_2\\ u_3\end{bmatrix} + \begin{bmatrix}0\\ 0\\ 0\\ n_4\\ 0\\ 0\end{bmatrix}[m_1] \qquad (9)$$

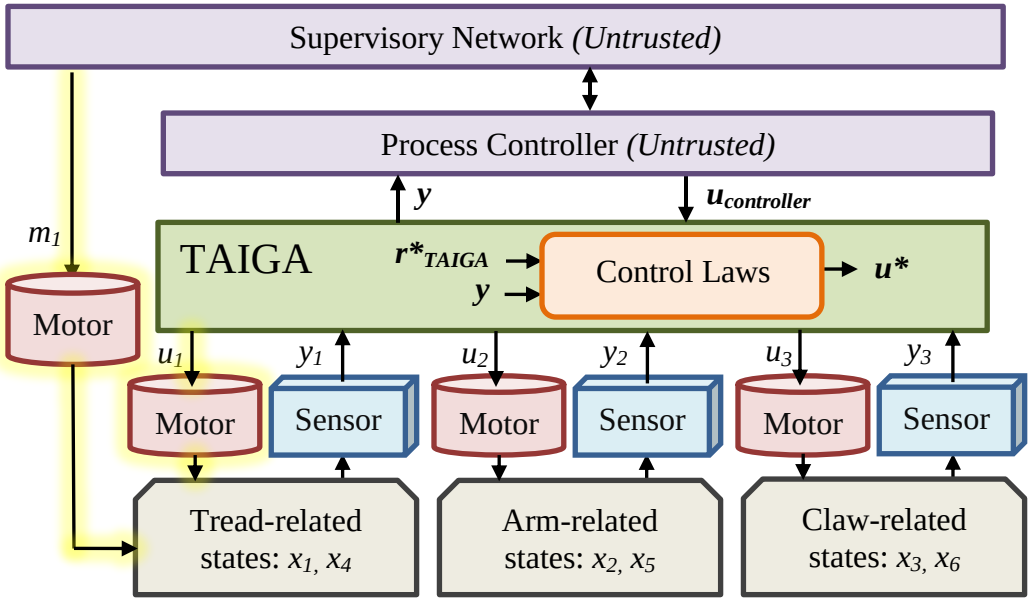

Figure 18: Configuration with competing untrusted and trusted actuators

We next consider a modified version of the previous configuration in Figure 18. The revised setup calls for multiple motors inside the tread, with one motor connected to TAIGA, and the other to the supervisory network. This setup may be common in plants that require high availability and use multiple redundant devices. The state-space matrix is similar to the previous setup, however, TAIGA still has trusted control over all states, as shown in equation (10). To maintain safety, however, TAIGA's trusted actuators must be able to negate the effects of the malicious actuators. Essentially, safety can be

maintained if ∃ a feasible $\boldsymbol{u}(t)$ s.t. $\boldsymbol{x}(t) \in \boldsymbol{S}(t)$ ∀ feasible $\boldsymbol{m}(t)$. If this condition is met, the configuration can satisfy *Automatic Safety Maintainability*. Similarly, the system satisfies the requirements for *Automatic Process Maintainability* iff ∃ a feasible $\boldsymbol{u}(t)$ *s.t.* $|\boldsymbol{x}(t) - \boldsymbol{x^*}(t)|$ < the maximum expected process error ∀ feasible $\boldsymbol{m}(t)$ and t. Special attention should be given to the feasibility constraint, since the actuators may quickly saturate while trying to counter-act one another. Furthermore, even though the system's safety or process may be maintained, significant additional energy may be consumed by the competing actuators. The criteria for maintainability have no implication for power efficiency unless power consumption is represented by a state variable.

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\\ \dot{x}_3\\ \dot{x}_4\\ \dot{x}_5\\ \dot{x}_6\end{bmatrix} = \begin{bmatrix}0 & 0 & 0 & a_{14} & 0 & 0\\ 0 & 0 & 0 & 0 & a_{25} & 0\\ 0 & 0 & 0 & 0 & 0 & a_{36}\\ 0 & 0 & 0 & 0 & 0 & 0\\ 0 & a_{52} & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 0\end{bmatrix}\begin{bmatrix}x_1\\ x_2\\ x_3\\ x_4\\ x_5\\ x_6\end{bmatrix} + \begin{bmatrix}0 & 0 & 0\\ 0 & 0 & 0\\ 0 & 0 & 0\\ b_{41} & 0 & 0\\ 0 & b_{52} & 0\\ 0 & 0 & b_{63}\end{bmatrix}\begin{bmatrix}u_1\\ u_2\\ u_3\end{bmatrix} + \begin{bmatrix}0\\ 0\\ 0\\ n_4\\ 0\\ 0\end{bmatrix}[m_1] \quad (10)$$

5.3.4 Operator Alarm & HMI Placement

The configuration in Figure 19 is very similar to the original setup, showing all sensors and actuators connected directly to TAIGA. The difference is that Figure 19 features an alarm, implemented using a red LED, and is locally connected to TAIGA. Because the alarm is only accessible through TAIGA and is a simple device, it is assumed to be protected from cyber-attacks. Because the system is again completely observable and the reference generator is internal to TAIGA, TAIGA is aware if the system is deviating from the programmed process. Because TAIGA can raise an external alarm, we can say the system features *Process Alarmability* and *Safety Alarmability***.**

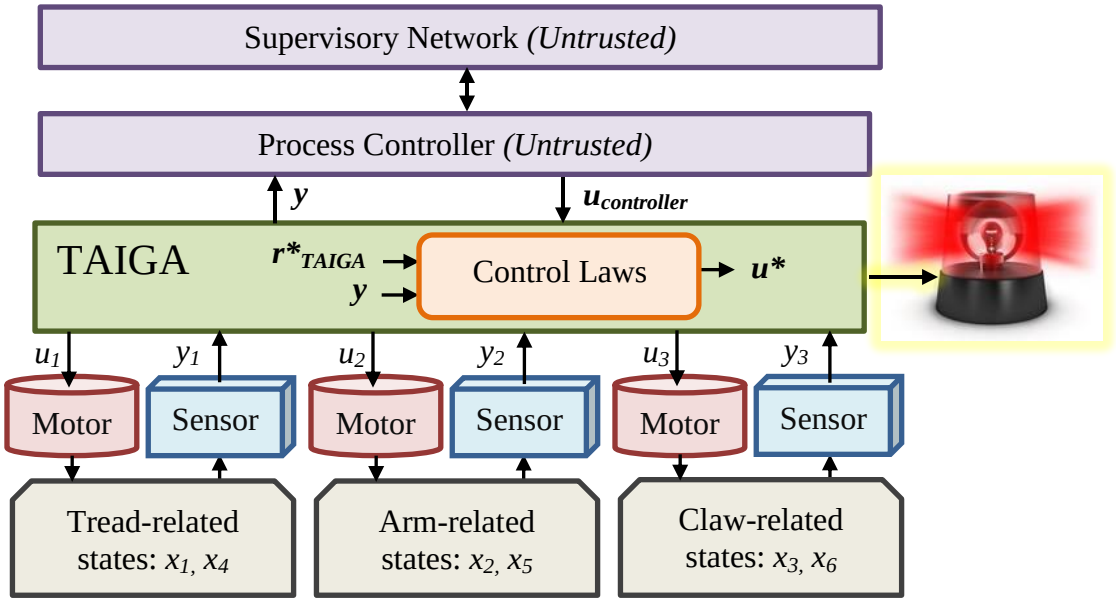

Figure 19: Configuration able to raise a local visual trusted

The configuration in Figure 20 again features a subtle change where the control laws have slightly changed. Whereas the previous setup was completely autonomous, the new setup features process reference signals unknown to TAIGA. For example, while the previous robot setup continuously retrieved and delivered cargo from the same stations, the setup in Figure 20 may instead be programmed to wait for a remote operator to drive the robot to a previously unknown pick-up station. This configuration resembles many remotely operated devices, such as surgical robots used to perform tele-operated surgery [70]. The process reference signals $\boldsymbol{r^*(t)}$ is now dependent on external system knowledge, and is not known internally to TAIGA. In this case, the process is dependent on the external $\boldsymbol{r}_{controller}(t)$, which in the event of cyber-attack, may be untrusted. Since the system is still completely observable and controllable within TAIGA, it is still *Automatic*

Safety Maintainable, though it lacks sufficient internal information to autonomously maintain the process.

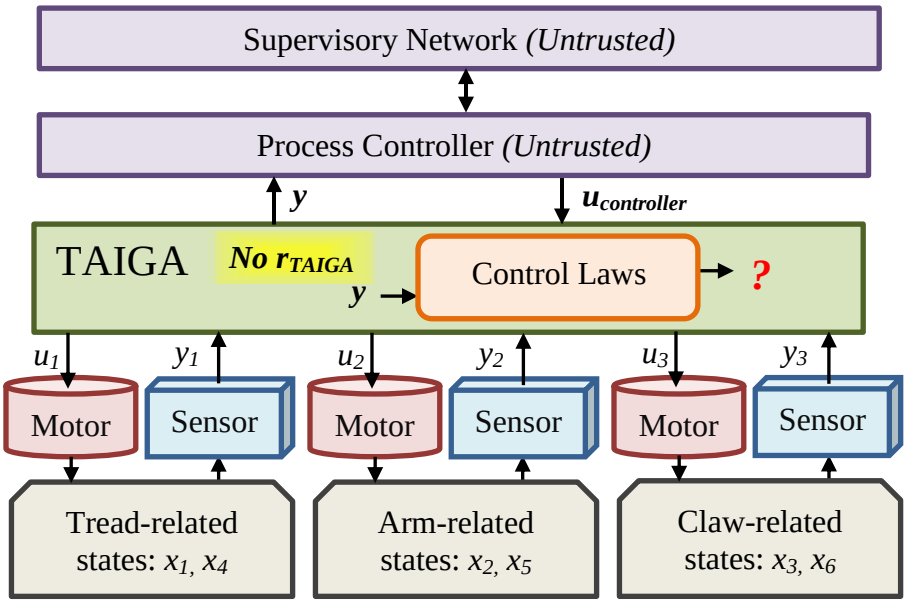

Figure 20: Configuration lacking process knowledge and therefor process protection

Another final configuration is shown in Figure 21. This configuration is still unable to internally generate the process reference signal $\boldsymbol{r^*(t)}$, however, TAIGA is equipped with a local analog HMI not networked to another device. Even in the absence of local operator commands $\boldsymbol{u_{\_HMI}(t)}$, the process can still be remotely operated with guaranteed *Automatic Safety Maintainability*. If a local operator were to provide a trusted input $\boldsymbol{u_{\_HMI}(t)}$, which differed from the untrusted $\boldsymbol{u_{\_Controller}(t)}$, TAIGA would select the trusted signal $\boldsymbol{u_{\_HMI}(t)}$. Similarly, if $\boldsymbol{u_{Controller}(t)}$ halted the process by attempting to drive to an unsafe state, the process could be safely resumed if a local operator physically accessed the simple HMI and issued $\boldsymbol{u_{\_HMI}(t)}$. We can therefore say the setup is eligible

for *Manual Process Maintainability*. It should be noted, however, that although $\boldsymbol{u\_HMI(t)}$ comes from a local operator, it still must undergo the same specification and operational guards as $\boldsymbol{u\_Controller(t)}$.Therefore even though the HMI requires physical access, it does not provide a bypass to the TAIGA safety protections.

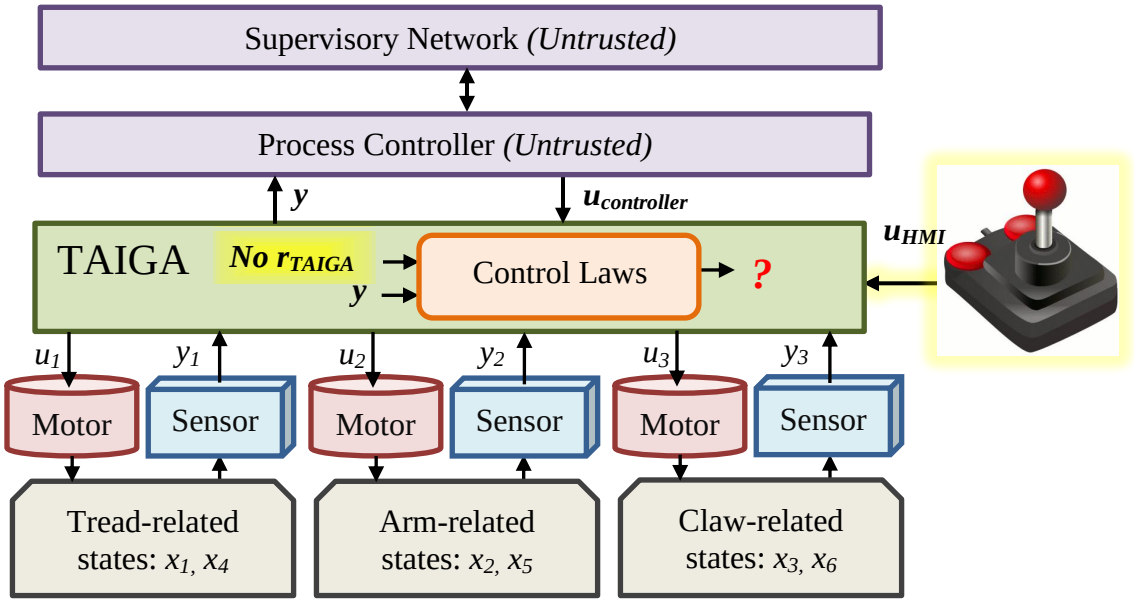

Figure 21: Configuration using a local human interface for process knowledge

5.4 TAIGA Benefits Eligibility Criteria

In the preceding sections, minor modifications to the setup architecture resulted in various different effects to the list of eligible benefits. In some setups, benefits could only be received if the state dynamics, control laws, and components satisfied certain conditions. Although an exhaustive list of all possible system configurations was not presented, most other configurations can be described using a combination of the discussed systems. As a result, the possible benefits provided by TAIGA can be

determined for any LTI CPS using the combinational logic circuit presented in Figure 22. In the figure, the actuator conditions are represented by tan inputs, the sensor and backup controller trigger switch conditions are represented by blue inputs, and control law and HMI conditions are represented by purple inputs. From the figure, it is clear that the various benefits depend on several system-level criteria which may not be immediately obvious. While the diagram does not account for all possible scenarios such as restrictions on the availability of $\boldsymbol{S}$, or setups involving a TAIGA system of systems, the chart provides important information about the suitability of TAIGA for many CPSes.

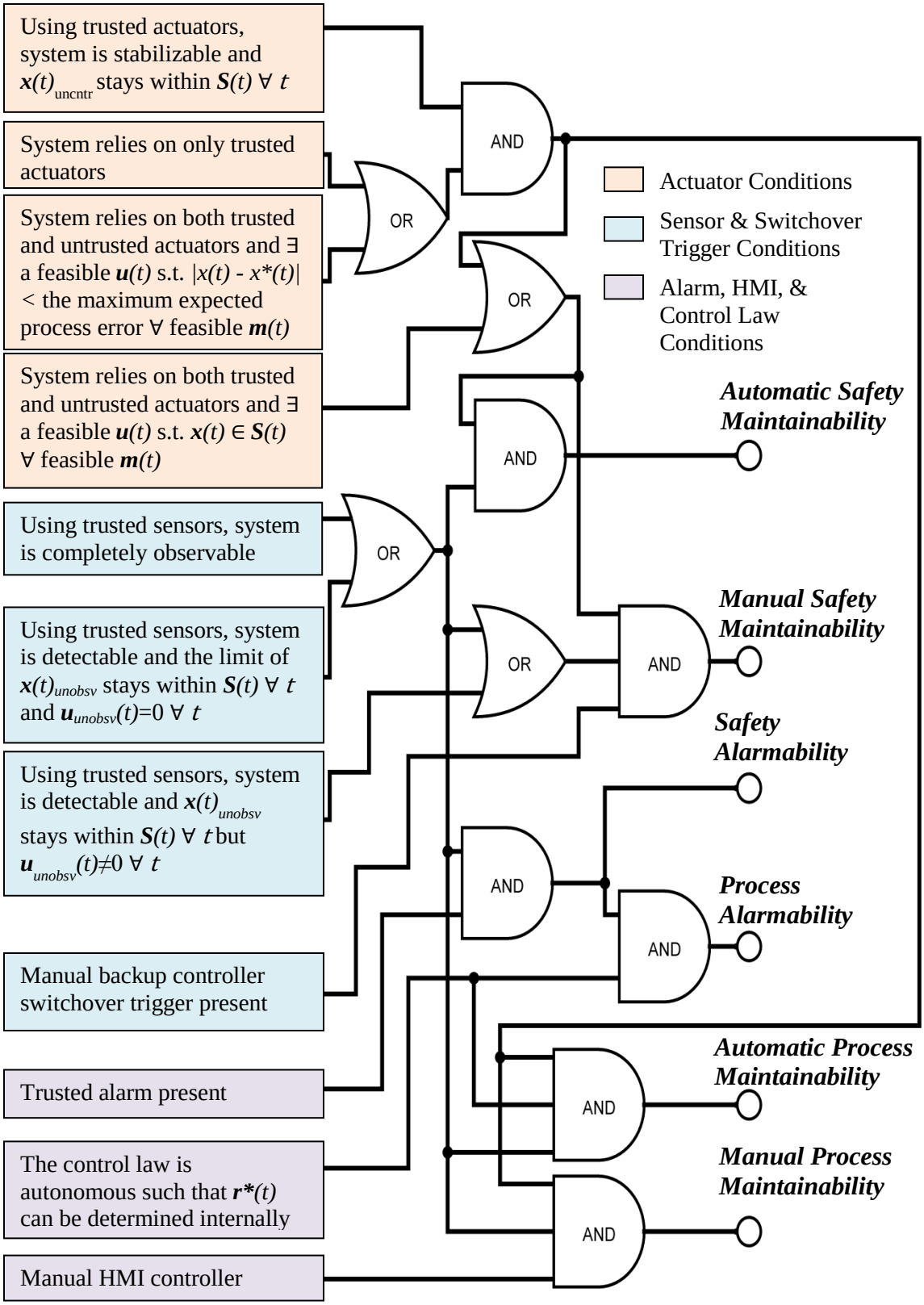

Figure 22: Logic diagram relating system properties to eligible TAIGA benefits

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

This thesis explored current CPS adversaries and cyber-attack vectors, and defined four possible attack outcomes on a process controller. The discussed TAIGA scheme can defend against these outcomes, while satisfying the trust requirements. The effectiveness of TAIGA was confirmed by the results of an implementation on a robot simulating cargo-carrying operations. Next, the benefits of TAIGA were categorized, and system eligibility criteria for each benefit were deduced by analyzing possible modifications the robot experiment. Finally, other CPS security techniques and previous TAIGA papers were discussed.

From the experimental evidence presented, it can be concluded that TAIGA can defend against the four cyber-attack outcomes in certain applications. In addition, this thesis showed that TAIGA may provide up to six different possible benefits to a CPS. Using the presented eligibility criteria to determine available benefits, TAIGA's suitability to any CPS can be easily decided before implementation. As applications of CPSes continue to grow, more solutions similar to TAIGA will be required to prevent cyber-attacks from inflicting physical harm.

Although TAIGA has evolved significantly since its introduction, the areas of additional future research and work are listed below:

- Additional implementations of TAIGA would provide useful experimental data regarding effectiveness, capabilities, and drawbacks. Possible platforms include more embedded systems, such as a motor controller on an industrial control system testbed. Another interesting implementation may be done using autonomous cars and aircraft, where obstacle avoidance and safety are a priority.
- TAIGA's scalability to large networks should be studied in greater detail from a system of systems perspective. Specifically, it should be determined what conditions must be met for a series of TAIGA protected devices to provide protection to an overall interconnected system.
- To meet the bandwidth requirements of systems involving a large number of components, larger-scale versions of TAIGA will need to utilize multiple parallel communication channels. Work will be needed to modify the specification guards to allow them to evaluate the effects of multiple simultaneous commands on a system.
- Current TAIGA prototypes are custom built, and as in the case of the robot experiment, use custom designs and communication protocols. If TAIGA is to be a cost-competitive security solution, significant work is required to reduce the implementation cost. This could be done by offering an easy to install version of TAIGA that interfaces with common communication adapters and protocols.

- A formal procedure for determining the safety and operational guards for any system should be established. This is a similar problem to determining the criteria for safety set ***S***.
- The current TAIGA benefits criteria are only applicable to LTI systems. Additional research will be required to expand the criteria to more complex systems.

APPENDIX A: PRIMARY CONTROLLER SOURCE CODE

Although the full C source code for the primary controller involves multiple files totaling several hundred lines of code, some sample functions are included here to represent the type of code written for this experiment. Figure 23 shows the low-level communication functions that handled messages to and from the SerDes communication queues. These include the bit-level functions such as *send_bits*, *request_bits*, and *get_bits*. Figure 24 shows the basic core functions which initialize and poll the sensors, and set servo positions. These functions include: *start_sensors*, *get_sensors*, and *rotate_servo*. Figure 25 shows the source code for an implementation of the robot's control state machine previously described in Figure 7. This section of code is a snippet from the main control loop, which runs continuously as the experiment is running. Figure 26 shows the *UI_command* function called by the main control loop whenever a command from the real-time user interface was detected. The UI only communicated with the controller via a string placed in a designed text file, so the controller was required to parse and execute the request. Possible UI inputs included: directly calling functions, changing program variables, or printing program variables.

```
//                          ********** Function: Send Bits ********************
//   Sends 32 bits message over the SerDes link to the plant. Commands are sent using the following form:
//  [4-bit command][4-bit device select ID][12-bit parameter #1][12-bit parameter #2]
int send_bits(unsigned int command_select, unsigned int device_select, unsigned int parameter_value1, unsigned int
parameter_value2){
int return_value;
//Validate that all parameter values are within acceptable ranges
if (command_select <= 15 && device_select <= 15 && parameter_value1 <= 4095 && parameter_value2 <=4095){
   //Shift and add bits in the correct order to create the 32-bit message
   u32 value_to_send =(command_select << 28)+(device_select << 24)+(parameter_value1 << 12)+(parameter_value2);
   //Add the value to the SerDes queue to be sent to the plant
   return_value = enqueue(value_to_send);
   if (return_value != 0) printf("RUNTIME ERROR: send_bits(), enqueue failure \n");
   return(return_value);
}
else
{
//If parameters are invalid, print a warning and prevent the values from being sent to the plant
printf("RUNTIME ERROR: send_bits(), values out of range \n");
return(1);
}
}

//                          ********** Function: Request Bits *************
//   Sends a 32 bit message to the plant requesting specific information. Request use the form:
//  [4-bits command = 0xD][7-bit tag response ID][21-bit sensor IO ID]
int request_bits(u32 device_ID, u32 message_id){
//check that all parameter values are within valid ranges
if (device_ID < 2097151 && message_id < 128){
   //shift and add bits to create a 32-bit message
   u32 value_to_send = (13 << 28) + (message_id << 21) + (device_ID);
   //Add the value to the SerDes queue to be sent to the plant
   enqueue(value_to_send);
return(0);
}
//If parameters are invalid, print a warning and prevent the values from being sent to the plant
else {printf("RUNTIME ERROR: get_bits(), device_ID out of range \n");}
return(1);
}

//                          ********** Function: Get Bits ********************
// Takes in u32 pointers for the message tag ID and data value. The function will "return" the message
// tag ID and value of the next message in the queue by placing them in the given pointers.
//Responses take the form: [4-bit = 0xF][7-bit tag ID specified in request_bits()][21-bit data]
int get_bits(u32 *message_id, u32 *value){
u32 rcv = 0;
//Check the incoming SerDes messages queue and return nothing if queue is empty
if (dequeue(&rcv, 0) != 0)return(1);
   //Remove the leading 0xF
   rcv = rcv - 0xF0000000;
   //Pull out the message tag ID
   *message_id = rcv >> 21;
   //Pull out the data value accompanying the tag
   *value = rcv - ((*message_id) << 21);
return(0);
}
```

Figure 23: Controller code showing low-level plant communication functions

```
//                              *************** Function: start_sensors() ******************
// Initializes the sensors by requesting information from each.
// Must be called at the start of the program or get_sensor() will not function properly.
int start_sensors(){
u32 rcv = 0;
//Wait for the SerDes queue to clear before sending
while (dequeue(&rcv, 0) == 0){;}
request_bits(RIGHT_LINE_DEV_ID, RIGHT_LINE_MSG_ID);                 // Request bits from the right line sensor
request_bits(LEFT_LINE_DEV_ID, LEFT_LINE_MSG_ID);                   // Request bits from the left line sensor
request_bits(BACK_RIGHT_LINE_DEV_ID, BACK_RIGHT_LINE_MSG_ID);//Request bits from the back right sensor
request_bits(BACK_LEFT_LINE_DEV_ID, BACK_LEFT_LINE_MSG_ID);   // Request bits from the back left sensor
printf( "Initalizing Sensors... \n");
return(0);
}


//                              *************** Function: get_sensor() ******************
int get_sensor(double *right, double *left, double *back_right, double *back_left){
// Takes in pointers to doubles representing the values of the current sensors.
// If a new value is available in the queue, the correct variable will be modified to
// reflect the new value. It will then request new data from the sensor which data was received.
int return_value = 0;
u32 message_id;
u32 value;
//Loop as long as there are incoming messages available
while (get_bits(&message_id, &value) == 0){
//Check is a specific sensor reading is available. If so, read it, and request the sensor to read again
if (message_id == RIGHT_LINE_MSG_ID){*right = value; request_bits(RIGHT_LINE_DEV_ID,RIGHT_LINE_MSG_ID);}
  else if(message_id == LEFT_LINE_MSG_ID){*left = value; request_bits(LEFT_LINE_DEV_ID, LEFT_LINE_MSG_ID);}
  else if(message_id == BACK_LEFT_ID){*back_left = value; request_bits(BACK_LEFT_ID, BACK_LEFT_ID);}
  else if(message_id == BACK_RIGHT_ID){*back_right = value; request_bits(BACK_RIGHT_ID, BACK_RIGHT_ID);}
return_value = 1;  // Indicates the loop ran sucessfully at least once
}
return(return_value);
}


//                ************** Function: Rotate Servo (GENERAL) ******************
// Called by other "Rotate" functions. Takes in the servo ID number, desired angle, and desired rate.
// Checks for valid values, and converts values to match the send_bits() notation
void rotate_servo(int servo_number, double angle, double rate_percent){
//Check and adjust for improper calls
if (angle>180) {angle = 180; printf("WARNING: Servo value over 180, setting to 180\n");}
if (angle<0) {angle = 0; printf("WARNING: Servo value under 0, setting to 0\n");}
if (rate_percent>100) {rate_percent = 100; printf("WARNING: Servo speed over 100%%, setting to 100%% \n");}
if (rate_percent<5) {rate_percent = 5; printf("WARNING: Servo speed under 5%%, setting to 5%% \n");}
//Convert message to values used by the servo controllers (500 = -90 deg, 2500 = +90 deg)
while (send_bits(11, servo_number, angle*(1000.0/90.0)+500, rate_percent*(30.0/100.0)) != 0);
}
```

Figure 24: General sensor and servo controller code functions

```
// STATE 1: Driving forward, looking to grab the cargo
if (state== RETRIEVING){
// If both front sensors read white, robot may be at the end of the track. Increment transition_tick.
      if (((right_line < S_THRESHOLD) && (left_line < S_THRESHOLD)) && new_data) transition_tick ++;
// If both front sensors don't read white, set transition tick back to 0.
      else if (new_data) transition_tick = 0;
// If transition_tick >= 2, the robot has read all white sensors multiple times, and is sure it is at the track's end.
// Requiring multiple sensor reads reduces the possibility of a false positive
      if (transition_tick >= 2 && state_change_permit){
                  transition_tick = 0;
                  // Perform the servo operations associated with transitioning states.
                  ang_vel = 90; set_angular_velocity(ang_vel, 100);
                  lin_vel = desired_reverse_vel;
                  set_linear_velocity(lin_vel, 100);
                  rotate_left_claw(180, 100);
                  rotate_right_claw(180, 100);
                  rotate_left_shoulder(180, 100);
                  rotate_right_shoulder(180, 100);
                  state= DELIVERING;
      }
}

// STATE 2: Driving backwards, looking to deliver  the cargo
if (state== DELIVERING){
// If both back sensors read black, robot may be at the end of the track. Increment transition_tick.
      if (((back_right_line >= THRESHOLD_UPPER) && (back_left_line >= THRESHOLD_UPPER)) && new_data)
transition_tick ++;
// If both front sensors don't read black, set transition tick back to 0.
      else if (new_data) transition_tick = 0;
// If transition_tick >= 3, the robot has read all black sensors multiple times, and is sure it is at the track's end.
// Requiring multiple sensor reads reduces the possibility of a false positive
      if (transition_tick >= 3 && state_change_permit){
                  transition_tick = 0;
                  // Perform the servo operations associated with transitioning states.
                  ang_vel = 90; set_angular_velocity(ang_vel, 100); lin_vel = desired_forward_vel;
set_linear_velocity(lin_vel, 100);
                  rotate_left_claw(45, 100);
                  rotate_right_claw(45, 100);
                  rotate_left_shoulder(90, 100);
                  rotate_right_shoulder(90, 100);
                  state= RETRIEVING;
      }
}
```

Figure 25: Main controller code showing the robot's action state machine

```
//                ********************* Function: UI_Command *************************
// Reads in the user's input string, parses it, and calls a function if syntax is valid
int UI_Command(char input_str[MAX_BUF], int *state, int *timer, int *exit_tick, int *ang_vel, int *lin_vel,
      int *desired_forward_vel, int *desired_reverse_vel,     int *new_data, int *transition_tick,
      double *left_line, double *right_line, double *back_left_line, double *back_right_line){
//par, par2, par3, are temporary variables which containe parsed sections of the input string
char * par;  char * par2;   char * par3;
int command = 0; //command corresponds to the ID number of which command (ex rotate _torso() ) will be called
double input_val1 = 0; //temporary variables to store converted strings to double parameter values
double input_val2 = 0;

par = strtok (input_str, " ,.-"); // parse the input string and store the result in par
// compare the first parameter with valid command calls, and set the command variable if a valid match is found.
command = 0;
if(strcmp(par, "head")==0){command = 1;}
if(strcmp(par, "torso")==0){command = 2;}
                ⋮
if(strcmp(par, "set_var")==0){command =16;}
if(strcmp(par, "print")==0){command =17;}

// ***Parsing and Error Checking***
//Check that the first parameter matches a command, otherwise return error.
if(command == 0){printf("Invalid Command Format. For a list of commands, type \"help\" \n"); return(0);}
//Get the input second parameter
par2 = strtok (NULL, " ,.-");
//Check that the second parameter is not empty
if (par2 == NULL){printf("Invalid Command Format. For a list of commands, type \"help\" \n"); return(0);}
//Convert from string to number if the second parameter should be a number (commands 1-15)
if(command < 16){input_val1=atof (par2);}
//If the command has a third parameter (commands 1-13, 16), get the third parameter
if (command != 14 && command != 15 && command != 17){
      par3 = strtok (NULL, " ,.-");
      //Check that the third parameter is not empty:
      if (par3 == NULL) {printf ( "Invalid Command Format. For a list of commands, type \"help\" \n"); return(0);}
      //Convert the third parameter from string to number
      input_val2= atof (par3);
} //end third parameter if

// Call the correct command based on the input parameters
switch(command){
case 1 : rotate_head(input_val1, input_val2); break;
                ⋮
case 15 : set_angular_velocity(input_val1, 100); break;
case 16 : //for command 16, “set_var”, search through possible adjustable variables for a match
      if(strcmp(par2, "state")==0){printf("Setting state to %d \n", (int)input_val2); *state = input_val2; break;}
                                ⋮
      else if(strcmp(par2, "desired_reverse_vel")==0){printf("Setting desired_reverse_vel to %d \n", (int)input_val2);
*desired_reverse_vel = input_val2; break;}
// For case 17, “print,”  print all known variable values to the terminal
case 17 : printf("Program Values: state %d, timer %d, exit_tick %d \n", *state, *timer, *exit_tick,);
                                ⋮
printf("RIGHT_LINE: %f   LEFT_LINE: %f   \n", *right_line, *left_line); break;
}
}
 return (EXIT_SUCCESS);
}
```

Figure 26: User interface function code called by the main controller

APPENDIX B: OTHER TAIGA EXPERIMENTAL SETUPS

During the final experiment's development, two other main setups were tested. These experiments included a flag-waving setup, and a variation of the hazardous cargo-carrying setup.

B.1 Flag Waving Setup

The flag waving experiment was a simple setup where the robot remained stationary while holding a "good" and "bad" flag. The primary controller allowed a user to send commands to wave the good flag. The user could also, however, activate malware that would sporadically send commands to raise the bad flag. TAIGA would prevent such actions, and would illuminate a red LED to alert the user of the attack. In this situation, malicious commands were rejected by TAIGA, however, they did not automatically trigger the backup controller. This is because TAIGA still required process information from the primary controller as to when the good flag should be raised. To test the backup controller, a switch was interfaced directly with TAIGA. When triggered, this switch invoked the backup controller, which would raise the good flag at pre-determined intervals, regardless of the user's requests.

B.2 Variation of the Hazardous Cargo Delivery Setup

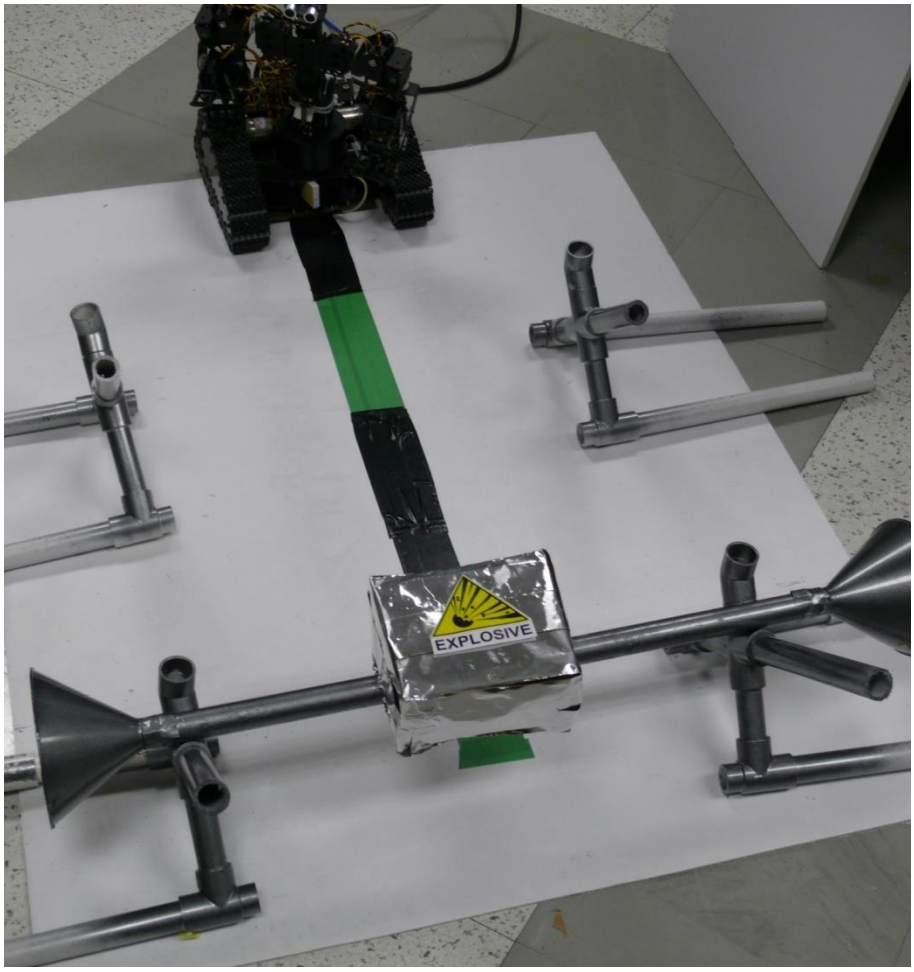

Figure 27: Variation of the hazardous cargo delivery setup

Figure 27 shows a variation of the hazardous-cargo delivery experimental setup. The experiment is similar to the one discussed in Chapter 4, however, it uses a different station mechanism, two colors of tape, and different control laws. Instead of using a ramped return mechanism, the cargo had to be carried back and forth between both stations. The simulated hazardous cargo consisted of a 5x5x6 inch box with a 16 inch piece of 1.5” PVC pipe running through the center. The PVC provided a gripping surface

for the robot's arms. Funnels were attached to both ends of the PVC pipe, which properly centered the cargo when dropped at each station. The stations were built using painted PVC as shown in Figure 28. To provide positioning information, green and black tape markings were arranged in four zones.

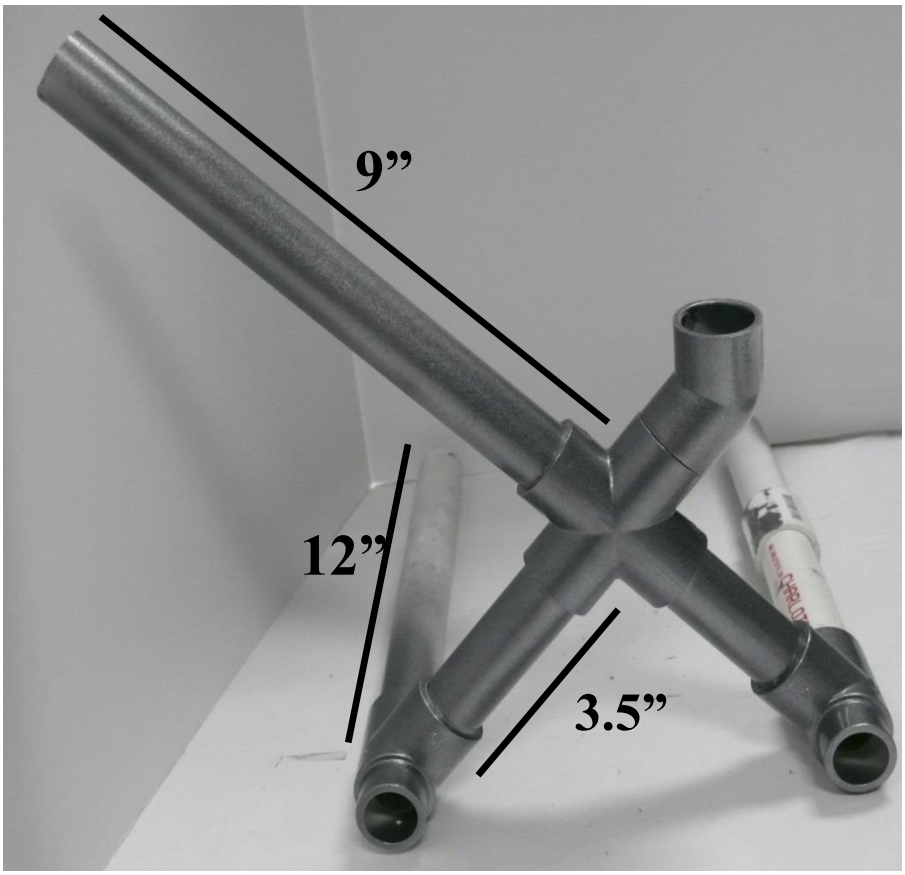

Figure 28: Cargo drop station

Because the robot preforms a series of complex actions in a sequence, the robot utilized different sets of control laws in each step. The set of control laws, which determines robot's direction, line-following algorithm, and arm positioning, was determined by the value of a status variable. The status variable follows a state machine that describes the robot's behavior with respect to the overall process as shown in Table 6. The status variable may be incremented by a floor marking color transition, which

indicates the robot has moved into a new physical zone, or by a timer, which indicates the time required for a specific action is complete.

Table 6: Robot control law at each status point

Status		**Control Law**			**State Information**	
Variable	Action	Tracking	Hands	Direction	Location	State Transition Trigger
0	Waiting for signal to start	None	*Don't Care*	Stopped	Rear Black	Network Signal
1	Grab Cargo from Station 1	Forward	*Don't Care*	Forward	Rear Black	Green Tape
2		Forward	*Don't Care*	Forward	Mid Green	Black Tape
3		None	Ready to Grab	Stopped	Forward Black	Timer
4		Forward	Ready to Grab	Forward	Forward Black	Green Tape
5	Drop Cargo at Station 2	None	Grabbing	Stopped	Front Green	Timer
6		Reverse	Holding	Reverse	Front Green	Black Tape
7		Reverse	Holding	Reverse	Forward Black	Green Tape
8	Wait at starting location	None	Dropping	Stopped	Mid Green	Timer
9		Reverse	Locked Up	Reverse	Mid Green	Black Tape
10	Grab Cargo from Station 2	None	Ready to Grab	Stopped	Rear Black	Timer
11		None	Ready to Grab	Forward	Rear Black	Green
12		Forward	Ready to Grab	Forward	Mid Green	Black
13	Drop Cargo at Station 1	None	Grabbing	Stopped	Mid Black	Timer
14		Forward	Holding	Forward	Forward Black	Green Tape
15	Wait at starting location	None	Dropping	Stopped	End Green	Timer
16		Reverse	Locked Up	Reverse	End Green	Black Tape
17		Reverse	*Don't Care*	Reverse	Mid Black	Green Tape
18		Reverse	*Don't Care*	Reverse	Mid Green	Black Tape

In this setting, the backup controller returns the hazardous cargo to its original station, and resets the robot to its starting location. A red LED provides a physical indication that the backup controller has taken over the system. Control is only restored to the primary

control through a physical hardware reset of the Zybo board. Since the backup controller is a modified simpler version of the primary control, the backup controller required minimal design effort.

REFERENCES

[1] CPS Steering Group, "Cyber-physical systems executive summary," in *CPS Summit 2008*, Mar 2008.

[2] National Institute of Standards and Technology, "Strategic Vision and Business Drivers for 21st Century Cyber-Physical Systems," U.S. Department of Commerce, Jan 2013.

[3] C. Axelrod, "Managing the risks of cyber-physical systems," in *IEEE Systems, Applications and Technology Conference (LISAT)*, Long Island, May 2013.

[4] J. Slay and M. Miller, "Lessons learned from the Maroochy water breach," *Critical Infrastructure Protection*, vol. 253, pp. 73-82, 2007.

[5] R. Langner, "Stuxnet: Dissecting a Cyberwarfare Weapon," *Security & Privacy, IEEE*, vol. 9, no. 3, pp. 49-51, June 2011.

[6] R. M. Lee, M. J. Assante and T. Conway, "ICS CP/PE (Cyber-to-Physical or Process Effects) case study paper – German Steel Mill Cyber Attack," Sans ICS, Dec 2014.

[7] M. d. Cava, *Hack of connected car raises alarm over driver safety,* San Francisco : USA Today , July 2015.

[8] F. Pasqualetti, F. Dorfler and F. Bullo, "Control-Theoretic Methods for Cyberphysical Security: Geometric Principles for Optimal Cross-Layer Resilient Control Systems," *Control Systems, IEEE* , vol. 35, no. 1, pp. 110-127, Feb 2015.

[9] S. Bopardikar and A. Speranzon, "On analysis and design of stealth-resilient control systems," in *Resilient Control Systems (ISRCS), 2013 6th International Symposium on* , San Francisco, CA , Aug 2013.

[10] L. Lerner, *Trustworthy Embedded Computing for Cyber-Physical Control,* Blacksburg, VA: Ph.D dissertation, Dept. ECE., Virginia Tech, 2015.

[11] J. Depoy, J. Phelan, P. Sholander, B. Smith, G. Varnado and G. Wyss, "Risk assessment for physical and cyber attacks on critical infrastructures," in *Military Communications Conference, 2005. MILCOM 2005. IEEE,* Atlantic City, NJ, 2005.

[12] M. Cheminod, L. Durante and A. Valenzano, "Review of Security Issues in Industrial Networks," *Industrial Informatics, IEEE Transactions on*, vol. 9, no. 1, pp. 277-293, May 2012.

[13] A. Teixeira, *Toward cyber-secure and resilient networked control systems,* Ph.D. dissertation, KTH Royal Institute of Technology, 2014.

[14] X. Feng, T. Lu, X. Guo, J. Liu, Y. Peng and Y. Gao, "Security Analysis on Cyber-physical System Using Attack Tree," in *Intelligent Information Hiding and Multimedia Signal Processing, Ninth International Conference on*, Oct 2013.

[15] IEEE Council on Electronic Design and Automation, "Can We Trust the Chips of the Future?," *Design & Test of Computers, IEEE,* vol. 28, no. 5, pp. 96-103, Sept. 2011.

[16] M. Tehranipoor, S. H, X. Zhang, M. Wang, R. Karri, J. Rajendran and K. Rosenfeld, "Trustworthy hardware: Trojan detection and design-for-trust challenges," *Computer,* vol. 44, no. 7, pp. 66-74, July 2011.

[17] A. Cárdenas, S. Amin, B. Sinopoli, A. Giani, A. Perrig and S. Sastry, "Challenges for securing cyber physical systems," in *First Workshop on Cyber-physical Systems Security*, Virginia, July 2009.

[18] S. Kuvshinkova, "“SQL Slammer worm lessons learned for consideration," North Amer. Elec. Reliab. Council, Atlanta, GA, 2003.

[19] T. Kiravuo, M. Sarela and J. Manner, "Weapons against Cyber-Physical Targets," in *Distributed Computing Systems Workshops (ICDCSW), 2013 IEEE 33rd International Conference on*, Philadelphia, PA, July 2013.

[20] B. Zhu, A. Joseph and S. Sastry, "A Taxonomy of Cyber Attacks on SCADA Systems," in *Internet of Things (iThings/CPSCom), 2011 International Conference on and 4th International Conference on Cyber, Physical and Social Computing*, Dalian, Oct. 2011.

[21] L. W. Lerner, M. M. Farag and C. D. Patterson, "Run-time Prediction and Preemption of Configuration Attacks on Embedded Process Controllers," in *1st International Conference on Security of Internet of Things*, Kerala, India, Aug 2012.

[22] C. Bao, Y. Xie and A. Srivastava, "A security-aware design scheme for better hardware Trojan detection sensitivity," in *Hardware Oriented Security and Trust (HOST), 2015 IEEE International Symposium on*, Washington, DC, USA, May 2015.

[23] M. Merabti, M. Kennedy and W. Hurst, "Critical infrastructure protection: A 21st century challenge," in *Communications and Information Technology (ICCIT), 2011 International Conference on*, Aqaba, March 2011.

[24] A. A. Cárdenas, S. Amin and S. Sastry, "Research challenges for the security of control systems," in *Proc. 3rd USENIX Workshop on Hot*, 2008.

[25] A. A. Cárdenas, Z.-S. L. S. Amin, Y.-L. Huang, C.-Y. Huang and S. Sastry, "Attacks against process control systems: Risk assessment, detection, and response," in *Proc.*

6th ACM Symp. Inf., Comput. Commun. Security, 2011.

[26] P. Loh, G. Sabaliauskaite and A. Mathur, "Detecting injection attacks in linear time invariant systems," in *Cybernetics and Intelligent Systems (CIS), IEEE Conference on*, Manila, Nov 2013.

[27] G. Sabaliauskaite and A. Mathur, "Countermeasures to Enhance Cyber-physical System Security and Safety," in *Computer Software and Applications Conference Workshops (COMPSACW), 2014 IEEE 38th International*, Vasteras, July 2014.

[28] T. L. Crenshaw, E. Gunter, C. L. Robinson, L. Sha and P. R. Kumar, "The Simplex Reference Model: Limiting Fault-Propagation Due to Unreliable Components in Cyber-Physical System Architectures," in *28th IEEE International Real Time Systems Symposium*, Dec 2007.

[29] X. Wang, N. Hovakimyan and L. Sha, "L1Simplex: Fault-tolerant control of cyber-physical systems," in *Cyber-Physical Systems (ICCPS), 2013 ACM/IEEE International Conference on*, Philadelphia, PA, April 2013.

[30] D. Seto, B. H. Krogh, L. Sha and A. Chutinan, "Dynamic control system upgrade using the Simplex architecture," *Control Systems, IEEE*, vol. 18, no. 4, pp. 72-80, Aug 1998.

[31] M. Cheminod, I. Bertolotti, L. Durante, P. Maggi, D. Pozza, R. Sisto and A. Valenzano, "Detecting Chains of Vulnerabilities in Industrial Networks," *Industrial Informatics, IEEE Transactions on*, vol. 5, no. 2, pp. 181-193, May 2009.

[32] D. Reinelt and M. Wolframm, "Security in virtual automation networks," in *Emerging Technologies and Factory Automation, 2008. ETFA 2008. IEEE International Conference on*, Hamburg, Sept 2008.

[33] R. Song, L. Korba and G. Yee, "A Scalable Group Key Management Protocol," *Communications Letters, IEEE*, vol. 12, no. 7, pp. 541-543, July 2008.

[34] D. Choi, S. Lee, D. Won and S. Kim, "Efficient Secure Group Communications for SCADA," *Power Delivery, IEEE Transactions on* , vol. 25, no. 2, pp. 714-722, 2009.

[35] K. Xiao and M. Tehranipoor, "BISA: Built-in self-authentication for preventing hardware Trojan insertion," in *Hardware-Oriented Security and Trust (HOST), 2013 IEEE International Symposium on*, Austin, TX, June 2013.

[36] Y. Gilad, A. Herzberg and A. Trachtenberg, "Securing Smartphones: A µTCB Approach," *Pervasive Computing, IEEE*, vol. 13, no. 4, pp. 72-79, Oct 2014.

[37] P. Shuanghe and H. Zhen, "Design and Implementation of Portable TPM Device Driver Based on Extensible Firmware Interface," in *Multimedia Information*

Networking and Security, 2009. MINES '09. International Conference on, Hubei, Nov 2009.

[38] K.-J. Lin and C.-Y. Wang, "Using TPM to improve boot security at BIOS layer," in *Consumer Electronics (ICCE), 2012 IEEE International Conference on*, Las Vegas, NV, Jan 2012.

[39] Z. Wang, Y. Wang and K. Luo, "Trusted computing technology analyzing in NGSCB," in *Electronic and Mechanical Engineering and Information Technology (EMEIT), 2011 International Conference on* , Harbin, Heilongjiang, China, Aug 2011.

[40] M. Barbareschi, E. Battista, V. Casola and A. Mazzocca, "On the Adoption of FPGA for Protecting Cyber Physical Infrastructures," in *P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2013 Eighth International Conference on* , Compiegne , 2013.

[41] G. Bloom, B. Narahari, R. Simha, A. Namazi and R. Levy, "FPGA SoC architecture and runtime to prevent hardware Trojans from leaking secrets," in *Hardware Oriented Security and Trust (HOST), 2015 IEEE International Symposium on* , Washington, DC, May 2015.

[42] A. Aysu, C. Patterson and P. Schaumont, "Low-cost and area-efficient FPGA implementations of lattice-based cryptography," in *Hardware-Oriented Security and Trust (HOST), 2013 IEEE International Symposium on*, Austin, TX, June 2013.

[43] G. Sabaliauskaite and A. Mathur, "Design of Intelligent Checkers to Enhance the Security and Safety of Cyber Physical Systems," in *Computer Software and Applications Conference Workshops (COMPSACW), 2014 IEEE 38th International*, July 2014.

[44] M. Farag, L. Lerner and C. Patterson, "Thwarting Software Attacks on Data-Intensive Platforms with Configurable Hardware-Assisted Application Rule Enforcement," in *Field Programmable Logic and Applications (FPL), 2011 International Conference on*, Sept. 2011.

[45] L. W. Lerner, Z. R. Franklin, W. T. Baumann and C. D. Patterson, "Application-level Autonomic Hardware to Predict and Preempt Software Attacks on Industrial Control Systems," in *44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2014),* Atlanta, GA, June 2014.

[46] Z. Franklin, C. Patterson, L. Lerner and R. Prado, "Isolating trust in an industrial control system-on-chip architecture," in *Resilient Control Systems (ISRCS), 7th International Symposium on*, Aug 2014.

[47] L. W. Lerner, Z. R. Franklin, W. T. Baumann and C. D. Patterson, "Using High-level Synthesis and Formal Analysis to Predict and Preempt Attacks on Industrial

Control Systems," in *22nd ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA 2014)*, Monterey, CA, Feb 2014.

[48] A. Goldenberg, M. Gryniewski and T. Campbell, "AARM: A robot arm for internal operations in nuclear reactors," in *Applied Robotics for the Power Industry (CARPI), 2010 1st International Conference on*, Oct 2010.

[49] X. Dun, J. Yuan and L. Chen, "The Auto-docking System Design for the Fuel Loading Robot Used in Hazardous Environment," in *Robotics and Biomimetics, 2006. ROBIO '06. IEEE International Conference on*, Dec 2006.

[50] Q. Zhao, T. Wang, T. Zhang and J. Chen, "γ Ray irradiation test of control system of nuclear emergency rescue robot," in *Information Science and Technology (ICIST), 2014 4th IEEE International Conference on*, April 2014.

[51] C. Marché and Y. Moy, *The Jessie plugin for Deductive Verification*, INRIA Saclay - Télé-de-France and LRI, CNRS UMR 8623, 2013.

[52] B. Shah and B. Krishnamurthy, "Implementation of MPU for a Safe FreeRTOS Frame-work," *International Journal on Recent and Innovation Trends in Computing and Communication* , vol. 3, no. 5, May 2015.

[53] D. Qiu, Q. Wang and Y. Zhou, "Steady-state output controllability and output controllability of linear systems," in *Computational Intelligence and Industrial Applications, 2009. PACIIA 2009. Asia-Pacific Conference on*, Wuhan, Nov 2009.

[54] Y. Wu, "On the sylvester-type matrix difference systems and the controllability and observability of the time-invariant systems," in *Southeastcon, 2009. SOUTHEASTCON '09. IEEE*, Atlanta, GA, March 2009.

[55] Y.-W. Tseng, S.-K. Kwak and R. Yedavalli, "Stability, controllability and observability criteria for the 'reciprocal state space framework'," in *American Control Conference, 2003. Proceedings of the 2003* , June 2003.

[56] P. Minnerup and A. Knoll, "Testing autonomous driving systems against sensor and actuator error combinations," in *Intelligent Vehicles Symposium Proceedings, 2014 IEEE*, Dearborn, MI, June 2014.

[57] X. Wang, Y. Chen, C. Lu and X. Koutsoukos, "On Controllability and Feasibility of Utilization Control in Distributed Real-Time Systems," in *Real-Time Systems, 2007. ECRTS '07. 19th Euromicro Conference on*, Pisa, July 2007.

[58] G. Sabaliauskaite and A. Mathur, "Intelligent Checkers to Improve Attack Detection in Cyber Physical Systems," in *Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), 2013 International Conference on*, Beijing, Oct. 2013.

[59] M. Morisawa, K. Kaneko, F. Kanehiro, S. Kajita, K. Fujiwara, K. Harada and H. Hirukawa, "Motion Planning of Emergency Stop for Humanoid Robot by State Space Approach," in *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, Beijing, Oct. 2006.

[60] O. A. Harshe, N. T. Chiluvuri, C. D. Patterson and W. T. Baumann, "Design and Implementation of a Security Framework for Industrial Control Systems," in *International Conference on Industrial Instrumentation and Control, IEEE*, Pune, India, May 2015.

[61] N. T. Chiluvuri, O. A. Harshe, C. D. Patterson and W. T. Baumann, "Using Heterogeneous Computing to Implement a Trust Isolated Architecture for Cyber-Physical Control Systems," in *CPSS '15 Proceedings of the 1st ACM Workshop on Cyber-Physical System Security*, Singapore, April 2015.

[62] A. Cardenas, S. Amin and S. Sastry, "Secure Control: Towards Survivable Cyber-Physical Systems," in *Distributed Computing Systems Workshops, 2008. ICDCS '08. 28th International Conference on*, Beijing, June 2008.

[63] S. Lei and Y. Jianying, "Robust reliable tracking controller design against actuator faults for LPV systems with state feedback," in *Control Conference (CCC), 2010 29th Chinese*, Beijing, July 2010.

[64] J. Chen, W. Zhang and Y.-Y. Cao, "Robust reliable feedback controller design against actuator faults for linear parameter-varying systems in finite-frequency domain," *Control Theory & Applications, IET*, vol. 9, no. 10, pp. 1595 - 1607, 2015.

[65] C. Tricaud and Y. Chen, "Optimal trajectories of mobile remote sensors for parameter estimation in distributed Cyber-Physical Systems," in *American Control Conference (ACC), 2010*, Baltimore, MD, June 30 2010-July 2 2010.

[66] S. Bi and Y. J. Zhang, "Defending mechanisms against false-data injection attacks in the power system state estimation," in *GLOBECOM Workshops (GC Wkshps), 2011 IEEE*, Houston, TX, Dec 2011.

[67] R. Mitchell and I. Chen, "Effect of Intrusion Detection and Response on Reliability of Cyber Physical Systems," *Reliability, IEEE Transactions on*, vol. 62, no. 1, pp. 199-2410, March 2013.

[68] D. Garcia, R. Barber and M. Salichs, "Design and development of a wireless emergency start and stop system for robots," in *Informatics in Control, Automation and Robotics (ICINCO), 2014 11th International Conference on*, Vienna, Austria, Sept 2014.

[69] S. Gopinath, "Effectiveness of auxiliary system monitoring & continuous hydrogen scavenging operation on hydrogen-cooled generator at power plant," in *Energy and Environment, 2009. ICEE 2009. 3rd International Conference on*, Malacca, Dec.

2009.

[70] T. Bonaci, J. Herron, T. Yusuf, J. Yan, T. Kohno and H. J. Chizeck, *To Make a Robot Secure: An Experimental Analysis of Cyber Security Threats Against Teleoperated Surgical Robots*, May 2015.